UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

TRABAJO FIN DE MÁSTER

Análisis y comparación de extracción de características en señales de audio

Máster Universitario en Ingeniería de Telecomunicación

Autor: Isaac González González
Tutores: Ana María González Marcos y Francisco de Borja Rodríguez Ortiz

FECHA: SEPTIEMBRE 2019

ANÁLISIS Y COMPARACIÓN DE EXTRACCIÓN DE CARACTERÍSTICAS EN SEÑALES DE AUDIO

AUTOR: Isaac González González
TUTORES: Ana María González Marcos[1] y Francisco de Borja Rodríguez Ortiz[2]

Grupo de Aprendizaje Automático[1]
Grupo de Neurocomputación Biológica[2]
Dpto. de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
SEPTIEMBRE 2019

Resumen

Resumen

El reconocimiento automático del habla (*Automatic Speech Recognition* o ASR por sus siglas en inglés) es un tema candente a día de hoy. Los casos de uso de ASR son diversos, desde centralitas de llamadas que gestionan automáticamente el área del problema que afecta al usuario hasta casos más avanzados como los asistentes de voz. La tecnología actual permite desarrollar sistemas de ASR eficientes en entornos semánticos restringidos.

En este trabajo utilizaremos distintos extractores de características sobre señales de audio para posteriormente reconocer los fonemas presentes en frases habladas en lengua inglesa. Se estudiarán tanto extractores clásicos con parámetros de entrada predefinidos como extractores de características entrenados mediante aprendizaje no supervisado. Estos últimos infieren cuáles son las características más relevantes de las cuales están compuestas las señales. En el proceso de extracción de características no supervisado se propone una mejora del modelo basada en el reentrenamiento de los *bias*. Este nuevo enfoque reduce el error en la clasificación de fonemas.

Para la extracción de características clásicas se estudiarán extractores *Mel Frequency Cepstral Coeficients*. Éstos pueden considerarse el estándar a la hora de trabajar con señales de voz. En cuanto a las características obtenidas con aprendizaje no supervisado, se usarán máquinas de Boltzmann, *deep belief networks* y una combinación entre *Mel Frequency Cepstral Coeficients* y máquinas de Boltzmann.

Una vez extraídas las características, se estudiarán distintos clasificadores (redes neuronales recurrentes y modelos ocultos de *Markov*). En el caso de las redes recurrentes se medirá como influye el contexto fónico de la señal temporal en el error de clasificación de fonemas.

Palabras Clave

Máquinas de Boltzmann, extractores de características, *Mel Frequency Cepstral Coeficients*, *Long-Short Term Memory*, *Gated Recurrent Unit*, Modelos Ocultos de Markov, TIMIT, clasificación de fonemas, re-entrenamiento de bias, contexto fónico

Abstract

Automatic Speech Recognition (ASR) is a hot topic today. The cases of use of ASR are diverse, from call centers that automatically manage the area of the problem that affects the user to more advanced cases such as voice assistants. Current technology allows the development of efficient ASR systems in restricted semantic environments.

In this work we will use different feature extractors on audio signals to later recognize the phonemes present in phrases spoken in English language. Both classical extractors with predefined input parameters will be studied as well as feature extractors trained through unsupervised learning. The latter infer which are the most relevant features of which the signals are composed. In the unsupervised feature extraction process an improvement of the model is proposed based on the retraining of the *bias*. This new approach reduces the error in phoneme classification.

For the classical feature extraction, *Mel Frequency Cepstral Coefficients* will be studied. These can be considered the standard when working with voice signals. For unsupervised learning features, Boltzmann machines, *deep belief networks* and a combination of *Mel Frequency Cepstral Coefficients* and Boltzmann machines will be used.

Once the features are extracted, different classifiers will be studied (recurrent neural networks and hidden models of *Markov*). In the case of recurrent networks, it will be measured how the phonic context of the temporal signal influences the phoneme classification error.

Keywords

Boltzmann machines, features extractors, *Mel Frequency Cepstral Coeficients*, *Long-Short Term Memory*, *Gated Recurrent Unit*, Hidden Markov Models, TIMIT, phonemes classification, bias re-training, phonetic context

Agradecimientos

A mi familia aguantarme, a amigos por estar siempre ahí (mención especial a Ryan y Vaquerizo por haberse pasado todo este tiempo preguntando que si lo he acabado ya) y a mis tutores Ana y Paco por toda la ayuda prestada.

Por último, agradecer también a la Universidad Autónoma de Madrid por la financiación recibida para este proyecto.

Índice general

Índice de figuras

Índice de tablas

1 Introducción

1.1. Motivación del proyecto

El sistema auditivo está especializado en reconocer el habla de distintas personas o sonidos del entorno e identificarlos. Sin embargo, el reconocimiento de audio para un computador es una tarea compleja. Para este, un sonido no es más que una serie de números definidos dentro de un rango determinado por la codificación del formato del audio a una frecuencia de muestreo dada, por lo que no es trivial inferir que es lo que está sucediendo en dicho audio.

Dentro del procesamiento de audio, quizás el tema que más en boga está actualmente es el reconocimiento del habla o *speech recognition*. Varias compañías se están lanzando a ello, ofreciendo productos cada vez más sofisticados que hacen uso de diversas tecnologías que incorporan reconocimiento del lenguaje, como pueden ser Alexa de Amazon, Siri de Apple, Cortana de Microsoft, o Google Assistant de Google, por citar algunos de los más conocidos.

En este ámbito, se están realizando grandes avances en el área de inteligencia artificial aplicada al reconocimiento de habla. Con el desarrollo de cierto tipo de redes neuronales como las *Long Short Term Memory* (LSTM) [2] se ha mejorado en muchas tareas, aunque aún sigue habiendo sitio para algoritmos clásicos como los modelos ocultos de Markov (HMM por sus siglas en inglés) [3].

Es este trabajo se presentará un análisis de distintos algoritmos de extracción de características de sonido (Mel- Frecuency Cepstral Coeficients o Restricted Boltzmann Machines entre otros) [4] [5]. Una vez extraídas, se reducirán sus dimensiones para poder ser visualizadas y analizadas.

Estas características se aplicarán después sobre distintos clasificadores (redes neuronales recurrentes con distintos tipos de celdas). Estas técnicas en conjunto se emplearán sobre la base de datos de reconocimiento de habla TIMIT [6], con el objetivo de reconocer los fonemas contenidos en cada una de las 6300 frases de muestra que contiene.

1.2. Objetivos y enfoque

El objetivo fundamental de este trabajo es estudiar y analizar algunas de las principales técnicas y conocimientos necesarios para introducirse en la ciencia y tecnología del *speech recognition*.

En base a esta problemática, se considerarán los siguientes subobjetivos:

- El análisis exploratorio sobre la base de datos seleccionada para este trabajo.
- El estudio y análisis de algunos extractores de características de sonido sobre nuestra base de datos.
- El estudio y análisis de distintos clasificadores a partir de alguna de las representaciones mencionadas en el punto anterior.

1.3. Organización de la memoria

La memoria consta de los siguientes capítulos:

- En el capítulo 1, capítulo actual, se presenta la motivación y objetivos de este proyecto, junto con la organización de la memoria.
- En el capítulo 2 se empezará explicando algunos conceptos relacionados con la fonología y se presentará la base de datos que se utilizará en este trabajo. Después se explicarán distintos algoritmos existentes hoy en día de extracción de características para trabajar con señales de audio. Por último se explicarán algunos clasificadores para secuencias de señal aplicados al reconocimiento de fonemas.
- En el capítulo 3 se realizará un análisis exploratorio de los datos y se concretarán los extractores de características utilizados, así como la arquitectura de la red empleada para clasificar los vectores de características extraídos. Finalmente se presentará la arquitectura de la red elegida.
- En el capítulo 4 se representarán las características aprendidas por las máquinas de Boltzmann (es decir, qué aprende a buscar el extractor sobre la señal) y las características extraídas por todos los extractores. Además, se propondrá nuevo un método para el entrenamiento de las máquinas de Boltzmann empleadas. Después, se realizará un experimento para medir las capacidades en distintas variaciones sobre la red elegida. Finalmente se presentarán los resultados obtenidos con los extractores y clasificadores utilizados.
- En el capítulo 5 se presentarán las conclusiones finales de este trabajo y las posibles líneas de trabajo futuro.

2

Reconocimiento de fonemas. Estado del arte

2.1. Introducción. Sistemas de reconocimiento de habla

En este trabajo trataremos con el reconocimiento de fonemas como primer paso a un potencial sistema de reconocimiento automático del habla (*Automatic Speech Recognition* en inglés o ASR por sus siglas). Existen dos aproximaciones fundamentalmente para diseñar un sistema ASR: a nivel de palabra o a nivel de fonema. En este trabajo analizaremos y estudiaremos este segundo enfoque.

El reconocimiento a nivel de palabra suele entregar mejores resultados pero por contra, necesita una base de datos mucho mayor y en principio no puede reconocer palabras que no estén en su diccionario. El reconocimiento de fonemas en cambio, al no necesitar un diccionario permite reconocer cualquier palabra, ya que reconocerá las palabras por los fonemas que las componen. Además, tiene la ventaja de que son menos las clases (61 que normalmente se reducen a 39) que ha de clasificar, en comparación con el reconocimiento de palabra (donde puede haber miles de clases).

En este capítulo empezaremos repasando los tipos de fonemas que hay, en qué se distinguen, sus propiedades y cuáles son sus características. Seguidamente repasaremos algunos de los algoritmos que típicamente se usan en la actualidad para el tratamiento de señales de audio. Estos se dividen fundamentalmente en dos fases: extracción de características y clasificación. Dentro de cada uno de estos grupos exploraremos algunos ejemplos y los conceptos más importantes.

Además, se utilizará en este trabajo la base de datos TIMIT [6], una de las bases de datos más importantes para el estudio de fonemas.

2.2. Fonemas

2.2.1. Introducción

El campo de estudio encargado de los fonemas y su pronunciación se conoce como fonética. En esta sección estudiaremos algunos conceptos de fonética para comprender las señales que vamos a analizar en este trabajo.

Un fonema[1] puede definirse como la unidad mínima de voz hablada (*speech*). Los fonemas se pueden dividir fundamentalmente en dos clases: vocales y consonantes.

Los diferentes fonemas se producen dependiendo de cómo la boca, la garganta o la nariz modifican el aire que sale de los pulmones. Concretamente, este aire pasa por la tráquea y la laringe, donde están las cuerdas vocales. Estas cuerdas vocales pueden juntarse o separarse. Si están lo suficientemente juntas, vibrarán al pasar el aire y producirán sonido. La mayoría de los fonemas se producen de este modo, salvando algunos oclusivos como /p/ o /b/ que se producen después de un bloqueo de aire total.

La diferencia fundamental entre los fonemas categorizados como vocales y los categorizados como consonantes es que los consonantes bloquean todo o parte del aire, mientras que los vocales dejan pasar todo el aire. Además los consonantes suelen ser de duración más corta que los vocales. Existen también los fonemas semivocales, con características de ambos grupos, como por ejemplo el fonema /y/ o /w/.

2.2.2. Fonemas Consonánticos

Los fonemas consonánticos se pueden subcategorizar dependiendo del lugar de la articulación (es decir, de dónde se bloquea la mayor parte del aire) y de la manera de articular.

En cuanto al lugar de articulación, éstos podrán categorizarse en labiales, dentales, alveorales (el paladar justo detrás de los dientes), palatales, velares (el músculo que está detrás del paladar) o glotales (de glotis, el espacio entre las cuerdas vocales). Ver Figura 2.1.

En cuanto a la manera de articular, se distingue según si el bloqueo de aire es total o parcial.

- Si hay un bloqueo total de aire, seguido por una explosión de aire se conoce como oclusivo o *stop* (/p/, /d/, /t/...).
- Si el aire sale en su mayor parte por la cavidad nasal, producirá fonemas *nasales* (/n/, /m/, ...).
- Se llaman *fricativos* si el flujo de aire no se bloquea del todo, pero no sale limpiamente (/f/, /z/, /s/...)
- Por último se llaman *africativos* a aquellos fonemas fricativos que vienen precedidos de un *stop*(/ch/).

2.2.3. Fonemas Vocálicos

Dentro de los fonemas vocálicos, se pueden clasificar dependiendo de la altura del punto más alto de la lengua. Se distingue si este punto es al principio o al final de la boca y la forma de

[1]El texto en el que nos basaremos es el capítulo 7 de [7], en inglés. Por tanto, aunque trataremos de buscar fonemas compartidos entre inglés y español para las explicaciones, en la medida de lo posible, conviene pensar estos ejemplos por su pronunciación en inglés.

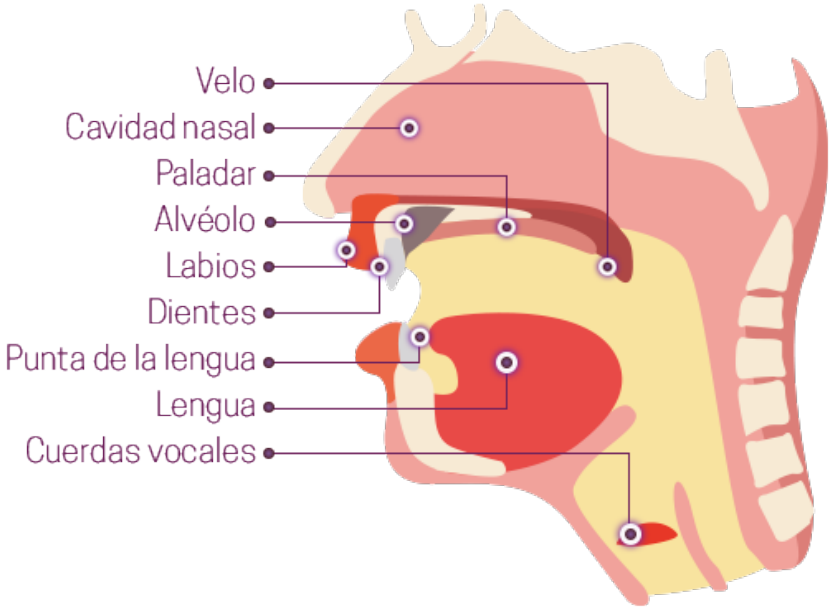


Figura 2.1: Sistema articulatorio de la voz humana. Imagen de la Universidad Nacional Autónoma de México, enlace: http://www.objetos.unam.mx/etimologias/consonantes/index.html.

los labios para pronunciar, como se muestra en la Figura 2.2. Además si la parte más alta de la lengua cambia de localización en el mismo fonema se conocen como diptongos, como por ejemplo pueden ser /ow/ o /aw/ en inglés.

También existen un grupo de fonemas que comparten características de vocálicos y consonánticos. Son cortos pero sonoros. Estos se conocen como semivocales (/w/ en *wish* o /y/ en *yell*).

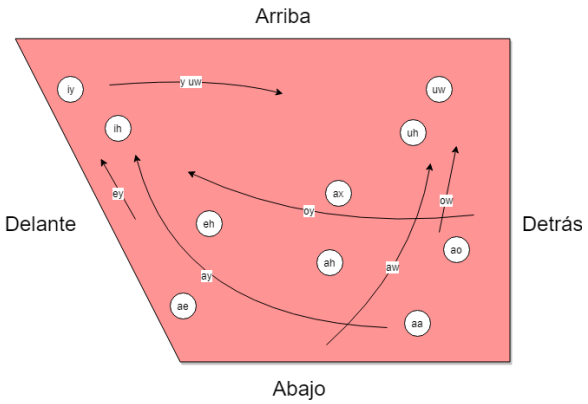


Figura 2.2: Posición de la parte más alta de la lengua para la pronunciación de vocales

Todas estas variaciones se traducen en distintas características de la señal. Cuantas más características compartan dos fonemas, más difícil será diferenciarlos, así pues, es de esperar que los fonemas /z/ y /s/ estén más cerca en un hipotético espacio de características que los fonemas /z/ y /aa/, por ejemplo. Además, los vocales al dejar pasar todo el aire limpio, tendrán componentes definidas más claramente en frecuencia, mientras que las características de los consonantes se verán más ruidosos.

2.3. Base de datos. TIMIT

Los modelos de *machine learning* requieren de una base de datos para ser entrenados. Según la naturaleza del proyecto, puede ser más o menos fácil encontrar bases de datos que se ajusten a ello. Por ejemplo, existen numerosas bases de datos de imágenes porque su etiquetado es relativamente sencillo, e incluso de audio para segmentación de palabras, pero no así para la segmentación de fonemas porque su etiquetado manual es tedioso.

En materia de reconocimiento de fonemas, TIMIT es de las bases de datos más utilizada, porque es una de las pocas que incluye etiquetado de fonemas de forma manual. Esta base de datos fue desarrollada en conjunto por Texas Instruments (TI), Massachusetts Institute of Technology (MIT) y SRI International (SRI) con el patrocinio de Defense Advanced Research Projects Agency - Information Science and Technology Office (DARPA-ISTO) en Octubre de 1990 [6]. Esta base de datos contiene 6300 frases en total de 630 hablantes distintos, 70 % hombres y 30 % mujeres, clasificados todos ellos en 8 regiones distintas de Estado Unidos (EE.UU.), con una distribución como se muestra en la Tabla 2.1. Estas regiones vienen marcadas por dónde pasó el hablante su infancia, con las excepciones de la región "Oeste de EE.UU." cuyas fronteras no están del todo claras y la región "Dialecto no definido" que se refiere a aquellos hablantes que no pasaron su infancia en un solo sitio.

En la Tabla 2.1 por cada región se presenta el número de hablantes de género masculino junto con el porcentaje que supone sobre el total de la región y el número de hablantes de género femenino junto con su porcentaje. Obsérvese que la suma de los dos porcentajes tiene que ser el 100 %. La última columna representa el total de hablantes de la región junto con el porcentaje que representan sobre el total de la muestra recogida en TIMIT.

Tabla 2.1: Distribución dialéctica de los hablantes en la base de datos de TIMIT

Dialect Region(dr)	**#Male**	**#Female**	**Total**
Nueva Inglaterra	31 (63 %)	18 (37 %)	49 (8 %)
Norte de EE.UU	71 (70 %)	31 (30 %)	102 (16 %)
Norte del interior de EE.UU	79 (77 %)	23 (23 %)	102 (16 %)
Sur del interior de EE.UU	69 (69 %)	31 (31 %)	100 (16 %)
Sur de EE.UU	62 (63 %)	36 (37 %)	98 (16 %)
Nueva York	30 (65 %)	16 (35 %)	46 (7 %)
Oeste de EE.UU	74 (74 %)	26 (26 %)	100 (16 %)
Dialecto no definido	22 (67 %)	11 (33 %)	33 (5 %)
	438 (70 %)	**192 (30 %)**	**630 (100 %)**

Cada una de estas frases es un archivo de audio mono en formato wav con un muestreo de 16 KHz y una resolución de 16 bits por muestra. Además, por cada archivo de audio, TIMIT posee dos archivos más, uno para palabras y otro para fonemas que marcan dónde empiezan y acaban cada palabra y cada fonema presente en el audio. Estas fronteras se definen con el número de muestra del archivo de audio, por tanto, la precisión de estas fronteras es de $\frac{1}{16\,KHz} = 0{,}0625\,ms$ (sin contar la precisión que haya tenido el etiquetador). La Figura 2.3 muestra un ejemplo de cómo se organiza y segmenta esta información por fonemas.

Además, la propia base de datos separa las frases en tres tipos:

- Tipo SA. Frases pensadas para resaltar las variantes dialécticas de los *speakers*, y que fueron leídas por todas las personas que participaron en las lecturas de las frases.
- Tipo SX. Frases fonéticamente compactas que se diseñaron para cubrir pares de fonemas y contextos fonéticos que fueran difíciles o de particular interés.

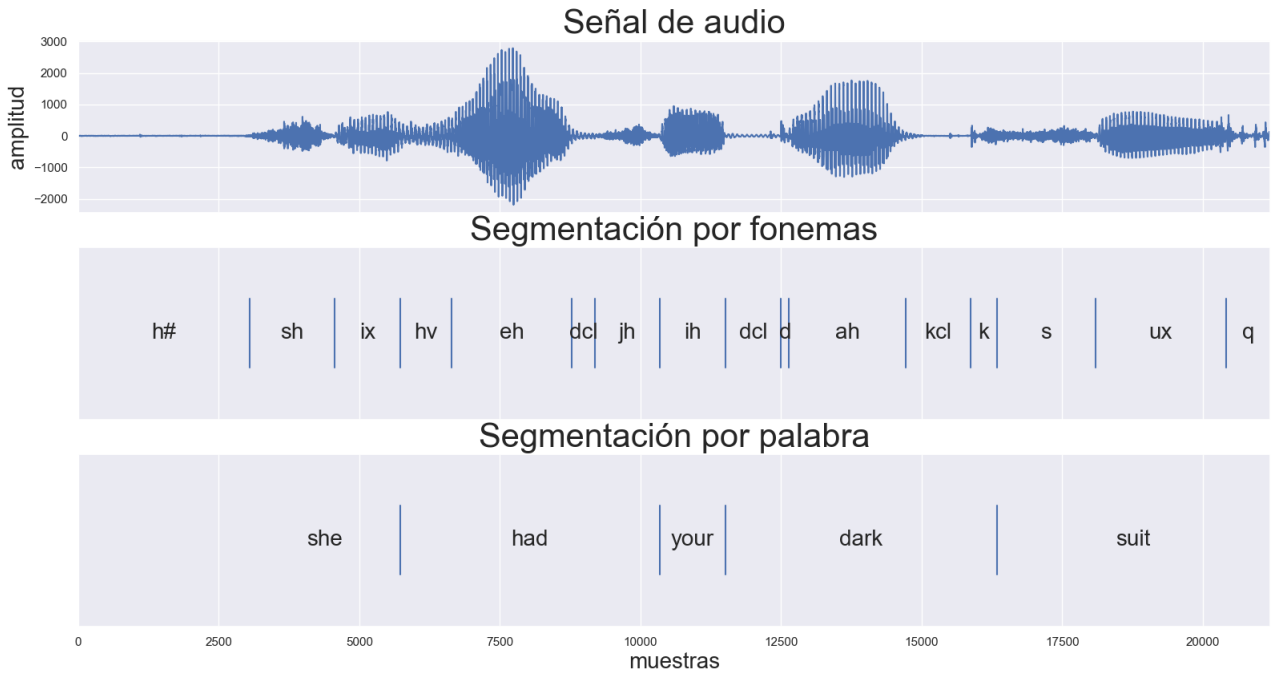


Figura 2.3: Visualización de la segmentación por fonemas y palabras de una señal de audio correspondiente a la frase *She had your dark suit*.

- Tipo SI. Frases fonéticamente diversas para añadir disparidad en cuanto a tipo de frases y contextos fonéticos.

La base de datos también incluye una separación sugerida entre archivos de *test* y de *train*, que es la que se utilizará en este trabajo. Con esta separación, los hablantes que aparecen en el banco de *train* no aparecen en el de *test* para evitar unos resultados demasiado optimistas. Por este mismo motivo, en muchos estudios que usan esta base de datos se excluyen también las frases *SA* del banco de *test* [8]. En este trabajo se procederá de igual modo.

En total, en TIMIT se reconocen 61 clases de fonemas distintos. Estos son: /b/, /bcl/, /d/, /dcl/, /g/, /gcl/, /p/, /pcl/, /t/, /tcl/, /k/, /kcl/, /dx/, /q/, /jh/, /ch/, /s/, /sh/, /z/, /zh/, /f/, /th/, /v/, /dh/, /m/, /n/, /ng/, /em/, /en/, /eng/, /nx/, /l/, /r/, /w/, /y/, /hh/, /hv/, /el/, /iy/, /ih/, /eh/, /ey/, /ae/, /aa/, /aw/, /ay/, /ah/, /ao/, /oy/, /ow/, /uh/, /uw/, /ux/, /er/, /ax/, /ix/, /axr/, /ax-h/, /pau/, /epi/ y /h#/.

Debido a la semejanza de muchos de ellos, es común reducir el número de fonemas de 61 a 39, agrupando para ello en el mismo fonema los que son similares (el fonema *q* se desecha) como muestra la Tabla 2.2. En este trabajo también se hará de este modo.

Tabla 2.2: Reducción de 61 fonemas (izquierda) a 39 fonemas (derecha). De los 61 fonemas, sólo se muestran los fonemas que han sido agrupados. El fonema /q/ se descarta.

/aa/, /ao/	/aa/
/ah/, /ax/, /ax-h/	/ah/
/er/, /axr/	/er/
/hh/, /hv/	/hh/
/ih/, /ix/	/ih/
/l/, /el/	/l/
/m/, /em/	/m/
/n/, /en/, /nx/	/n/
/ng, /eng/	/ng/
/sh/, /zh/	/sh/
/uw/, /ux/	/uw/
/pcl/, /tcl/, /kcl/, /bcl/, /dcl/, /gcl/, /h#/, /pau/, /epi/	/sil/
/q/	-

2.4. Preprocesado de audio.

2.4.1. Introducción

En aprendizaje automático, una característica es un propiedad individual y medible o rasgo de un fenómeno observable [9]. La extracción de características es una parte importante de los problemas de aprendizaje automático y puede tener un gran impacto en el resultado final del modelo.

Aunque con frecuencia estas características reducen dimensionalidad de la señal en crudo, la tarea más importante de un buen extractor de características es la de magnificar la información contenida en la señal que sea importante para el problema dado, y reducir en la medida de lo posible la que no es relevante.

Las técnicas de extracción de características más comúnmente usadas suelen ser **predefinidas**, como pueden ser la técnica de Mel-Frecuency Cepstral Coefficients (MFCC) [4], en contrapartida con las características extraídas con **aprendizaje no supervisado** [10]. Las primeras suelen dar buenos resultados y son más fáciles de extraer ya que no requieren un entrenamiento previo, sin embargo, las segundas cuentan con la ventaja de poder adaptarse, en principio, a cualquier tipo de señal. En este trabajo se explorarán algunas de estas técnicas, tanto las predefinidas como las aprendidas, comenzando primero con las tareas comunes a ambos, como son la fragmentación y el enventanado de la señal.

2.4.2. Fragmentación

La fragmentación y el enventanado son tareas comunes para la extracción de características independientemente de la técnica utilizada. Para trabajar con señales de audio es fundamental trabajar con secciones cortas de ellas ya que el espectro cambia muy rápidamente. Técnicamente, se dice que el *speech* es una señal no-estacionaria porque sus propiedades estadísticas no son constantes a lo largo del tiempo ya que cambian según la persona va hablando, sin embargo, fragmentando la señal en secciones de unos pocos milisegundos podemos asumir quasi-estacionaridad (es decir, considerar sus propiedades estadísticas constantes), lo que nos permitirá aplicar la transformada de Fourier, Fast Fourier Transform [11](FFT) por su rapidez, para obtener su espectro en frecuencia.

Aún con todo, suele ser necesario aplicar una función de ventana para atenuar las discontinuidades que se crean entre el final de un fragmento de señal y el principio del siguiente.

2.4.3. Enventanado de señal

Las transformadas de Fourier (FT) [2] (ver [12] capítulo 4) son herramientas matemáticas para expresar señales en función de exponenciales complejas. Es importante que no existan discontinuidades entre el final de una señal y el principio de ésta. En caso de que las hubiera, éstas serían aproximadas con sinusoides de muy alta frecuencia y que realmente no son parte de la señal. Estas sinusoides ensucian el espectro y pueden dar lugar a una menor tasa de acierto. Este fenómeno se conoce como **fuga espectral** [13].

La Figura 2.4 muestra el fenómeno de la fuga espectral, en el panel superior (a) se muestra una señal no periódica en el espacio de la ventana elegida (imagen de la izquierda) y su FFT en el dominio de frecuencias (imagen derecha). Como se puede observar aparece el pico más alto correspondiente a la frecuencia de la función sinusoidal y varios picos de menor amplitud a ambos lados del principal que simulan interferencias, ruido o fluctuaciones en la señal en el dominio del tiempo provocadas por el fenómeno de la fuga espectral. En el panel inferior (b) se muestra una señal con periodicidad en la ventana seleccionada (parte izquierda) y su representación de la FFT en el dominio de la frecuencia. Se puede observar que, en este caso, solo aparece un pico, el correspondiente a la frecuencia de la onda sinusoidal.

Para minimizar la fuga espectral, en la medida de lo posible, se enventanan los distintos fragmentos de la señal con **funciones de ventana**. Estas funciones de ventana suavizan la señal en los bordes, haciendo que la transición de un fragmento a otro sea lo más continua posible.

Una función ventana puede definirse por tres parámetros: el tamaño de la ventana, el *offset* o pasos entre ventanas sucesivas y la función (o forma) de la ventana.

Para seleccionar un tamaño adecuado de ventana, habrá que tener en cuenta ciertas consideraciones, ya que hacerla más grande o más pequeña tiene sus ventajas e inconvenientes.

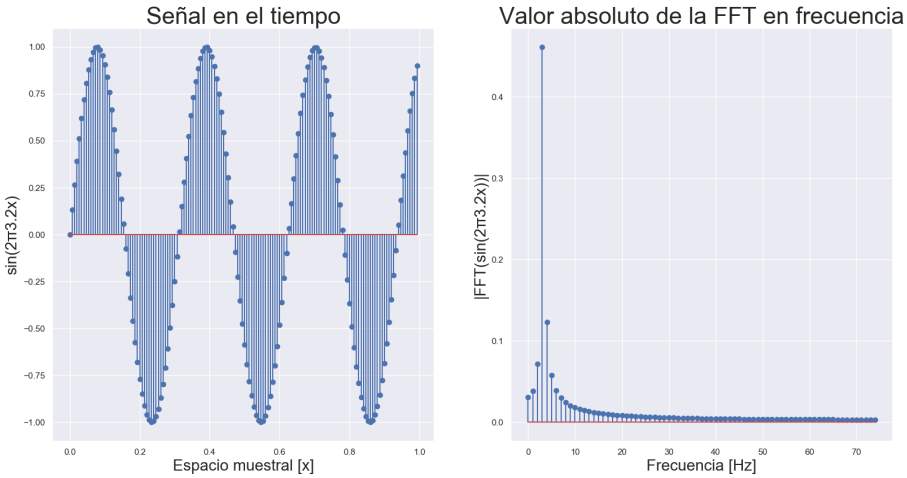
Por lo mencionado anteriormente en la sección 2.4.2, se sabe que se necesita poder asumir quasi-estacionaridad, por tanto, el tamaño de la ventana ha de estar restringido a unos pocos milisegundos. No se puede considerar quasi-estacionaria una señal de *speech* mucho mayor [14], por ello, cuanto más pequeño sea el tamaño de la ventana, más cierta será la asunción de estacionaridad. Sin embargo, con ventanas muy cortas también tendremos menor resolución en frecuencia (por el principio de incertidumbre de Fourier [15]), y además un mayor coste computacional al tener más ventanas a procesar.

Por todo esto, el tamaño típico para trabajar con *speech* es de entre 20 y 40 milisegundos [14] aproximadamente. En cuanto al *offset* entre ventanas sucesivas suele ser algo menor que la mitad del tamaño de la ventana, es decir, que se solapan parcialmente entre ellas para compensar la atenuación sufrida en los extremos al aplicar alguna función ventana como las que se explican en la siguiente sección.

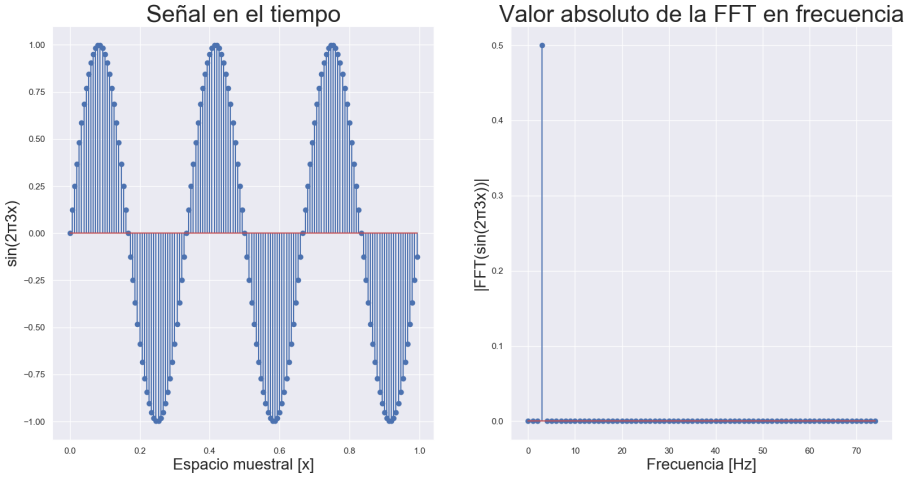
Funciones de ventanas

Existen varios tipos de funciones ventana y la principal diferencia entre ellas está en los lóbulos de su representación en frecuencia.

[2] se utiliza en realidad la Transformada de Fourier Discreta o DFT para computación porque los ordenadores manejan unidades discretas, ver [12] capítulo 5



(a) Señal con un número no entero de periodos completos dentro de la ventana. Debido a esto, se puede apreciar que su FFT no es limpia. Sólo se muestra la parte positiva del espectro.



(b) Señal con 3 periodos exactos. No existe fuga espectral en la FFT. Sólo se muestra la parte positiva del espectro.

Figura 2.4: Comparación de la fuga espectral de la FFT sobre una señal con un número no entero de periodos (a), y una señal con un número entero de periodos (b).

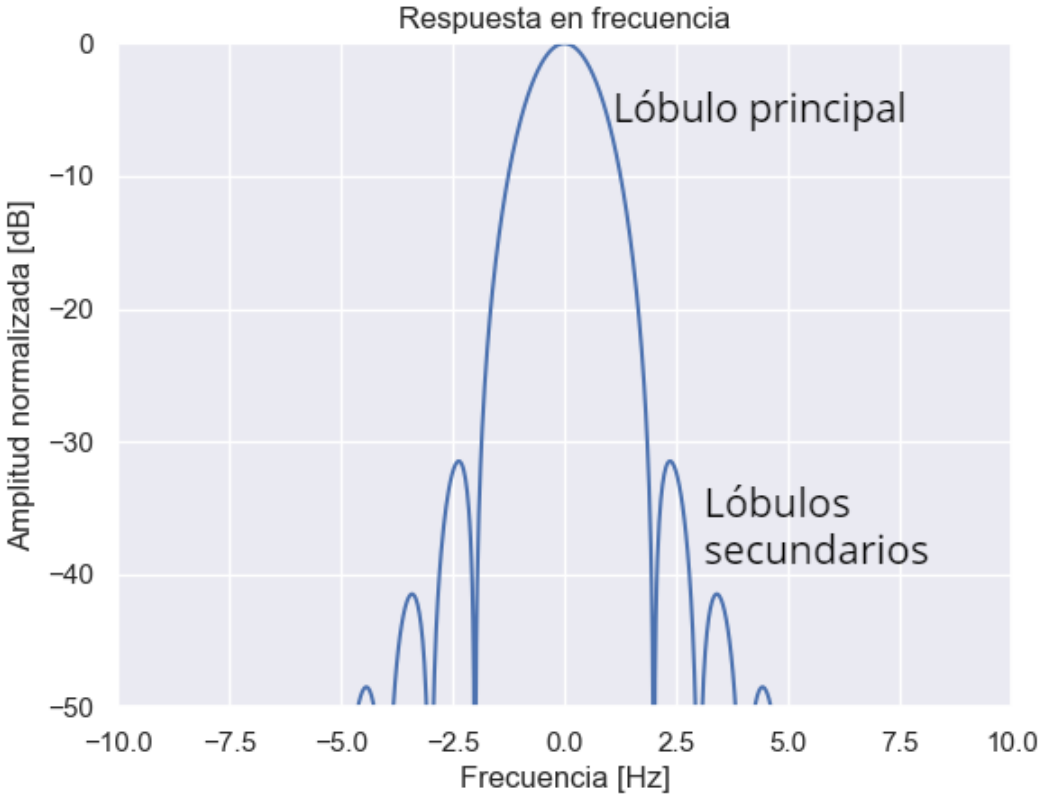


Figura 2.5: Identificación de los lóbulos en el espectro de una señal.

La representación en frecuencia de la función ventana nos da información sobre cuál será la resolución de la señal y las perturbaciones que sufre en el espacio de frecuencia. Más concretamente, cuanto más estrecho sea el lóbulo principal, mejor será la resolución en frecuencia y cuanto menores sean los lóbulos secundarios menor será la perturbación en frecuencia (ver Figura 2.5). Convendrá, por tanto, que estos lóbulos secundarios sean lo más bajos posible y que el lóbulo principal sea lo más estrecho posible. Lamentablemente no es posible tener una ventana con un lóbulo principal infinitamente estrecho y sin lóbulos secundarios, por tanto se trata de buscar un equilibrio entre estos dos conceptos.

La ventana más básica posible es la **ventana rectangular**. Esta se define como:

$$w[n] = \begin{cases} 1, & 0 \leq n \leq N-1. \\ 0, & \text{con otro valor,} \end{cases} \tag{2.1}$$

donde N es el tamaño de la ventana, y realmente sería como no aplicar ninguna función de ventana, por lo que no soluciona el problema de la fuga espectral. En este caso, la representación de la señal en el tiempo es perfecta, no hay distorsión (ver Figura 2.6.a) y su resolución en frecuencia es lo más precisa posible para el tamaño de la ventana (el lóbulo principal es relativamente estrecho), pero en frecuencia sufre graves perturbaciones debido a la magnitud de los lóbulos secundarios (ver Figura 2.6.b).

Típicamente, la función de ventana más usada es la ventana de Hamming (Figura 2.7). Esta reduce la fuga espectral debido a su atenuación en los extremos y además los lóbulos secundarios son más bajos, sin embargo tiene menor resolución en frecuencia ya que el lóbulo principal es ancho.

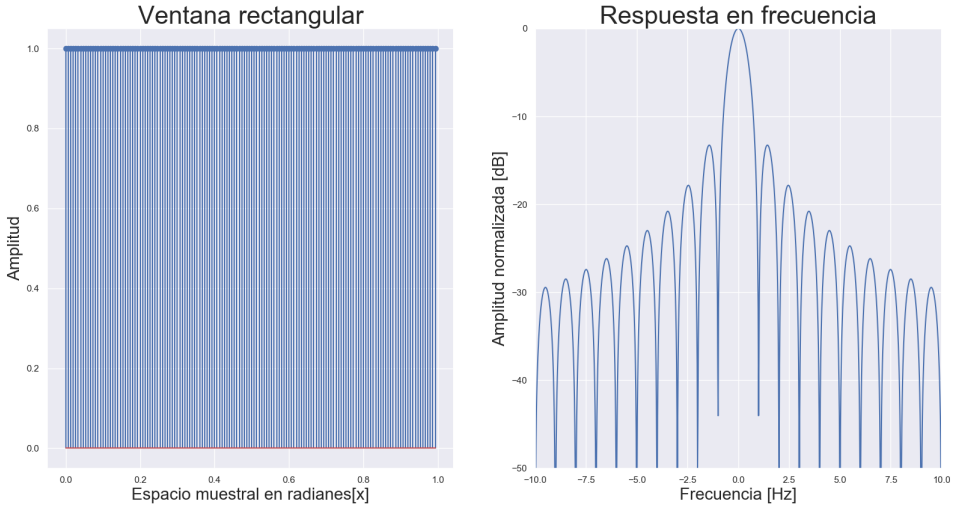


Figura 2.6: Ventana rectangular y su representación en frecuencia.

La función de ventana de Hamming se define como:

$$w[n] = a_0 - (1 - a_0)\, cos(\frac{2\pi n}{N-1}), \quad 0 \leq n \leq N-1, \tag{2.2}$$

donde N es el tamaño de la ventana.

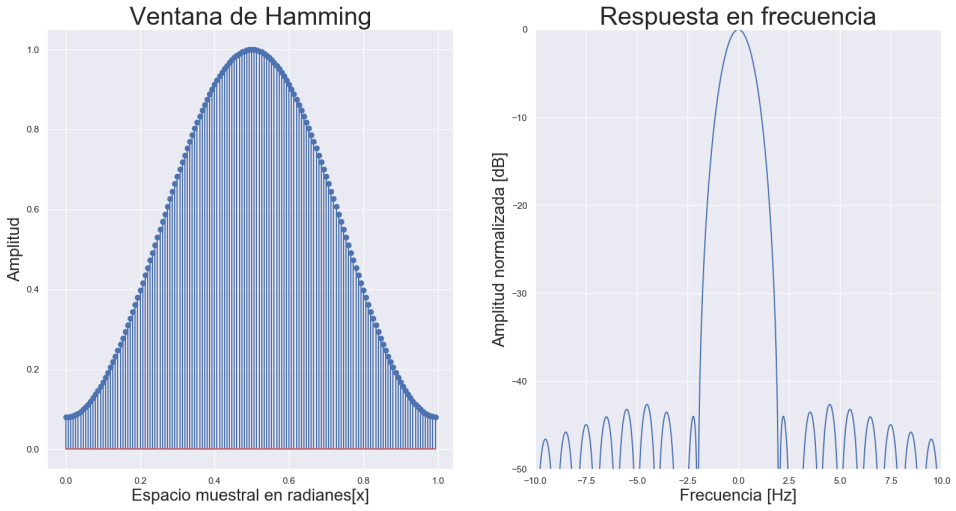


Figura 2.7: Ventana de Hamming y su representación en frecuencia. Se observa mejoría en la altura de los lóbulos secundarios con respecto a la ventana rectangular.

La ventana de Hann (Figura 2.8) es una particularización de la ventana de Hamming para $a_0 = 0{,}5$ y $a_1 = 0{,}5$. Se define como:

$$w[n] = (1/2)\, cos(\frac{2\pi n}{N-1}), \quad 0 \leq n \leq N-1, \tag{2.3}$$

donde N es el tamaño de la ventana.

Por supuesto, hay muchos más tipos de ventanas. Otras ventanas interesantes son la de Blackman [16] [17], Kaiser [18], etc.

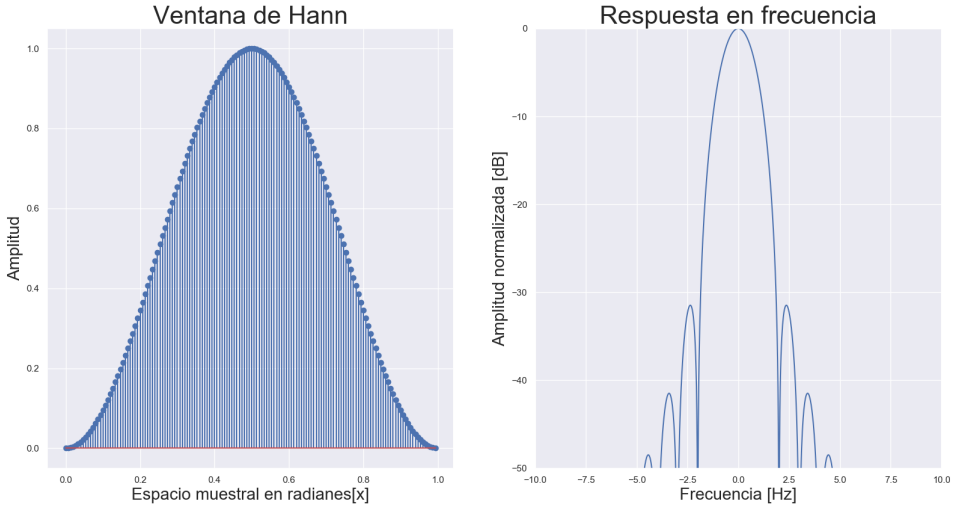


Figura 2.8: Ventana de Hann y su representación en frecuencia

2.5. Extracción de características predefinidas

Cuando se trata de reconocer patrones de la voz humana resulta interesante, y suele dar buenos resultados, el tratar de imitar el oído humano en la extracción de características, ya que éste está adaptado a reconocer patrones en nuestras voces. Esta es la línea que siguen muchos de los extractores de características predefinidos, biológicamente inspirados en muchos casos.

2.5.1. Escala de Mel

El objetivo de la escala de Mel es adaptar la linealidad objetiva de la frecuencia en Hz a otra subjetiva que el oído humano aprecie como lineal.

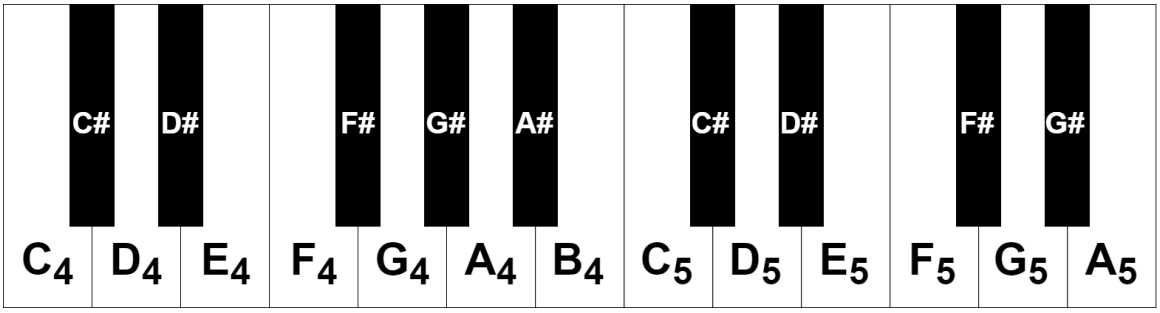


Figura 2.9: Representación de la escala musical en sistema anglosajón. La distancia entre cada tecla adyacente es de medio tono.

Físicamente los semitonos de la escala musical (ver Figura 2.9) no están equiespaciados en frecuencia. Esto es, definiendo la nota **La** (**A4** en el sistema anglosajón) en 440 Hz, el siguiente **La** en una octava superior (**A5**) estaría en dos veces esta frecuencia (880 Hz), sin embargo, los 12 semitonos que hay entre estas dos notas no está a la misma distancia en frecuencia, aunque para el oído pueda parecer que es así. Así por ejemplo subiendo un tono desde **A4** se encuentra el **Si** (**B4**) en los 493.88 Hz, y subiendo de nuevo otro tono desde este **B4** estaríamos en **Do#** (**C5#**) en 554.37 Hz. Se puede observar que entre el primer y el segundo tono de la escala hay

53.88 Hz, y entre el segundo y el tercero 60.49 Hz (en torno a un 12 % más), aunque para el oído humano pueda parecer la misma distancia.

En 1937 Stevens, Volkmann y Newman [19] propusieron una escala musical perceptual en los que estos semitonos estarían a una distancia semejante al oído humano, esta es la escala de Mel.

La escala de Mel toma como punto de partida que 1000 mels son 1KHz, y se define a partir de lo que los oyentes consideraban intervalos equiespaciados. Esta escala, por tanto se definió de un modo subjetivo.

Existen otras escalas menos subjetivas, como la escala de Bark [20] basada en las bandas críticas del oído, sin embargo la escala de Mel es la más popular en cuanto a reconocimiento de voz se refiere y aunque esta escala no suele ser considerada por si sola un extractor de características, ya que no discriminaría bien la mayoría de las entradas, es la base de la que parten mucho de los más populares extractores de características de voz, como los que se verán a continuación.

La fórmula de la escala de Mel se suele definir como 2.4, aunque existen otras variantes cambiando el logaritmo, etc:

$$m = 2595\, log_{10}(1 + \frac{f}{700}), \tag{2.4}$$

donde f es la frecuencia en herzios y m la correspondiente frecuencia en la escala de Mel. La transformación a Hz sería como en 2.5:

$$f = 700(10^{m/2595} - 1). \tag{2.5}$$

Su representación es como se indica en la Figura 2.10. Como se puede apreciar por la pendiente de la gráfica, esta escala ofrece más resolución en las bajas frecuencias y menos resolución en las altas.

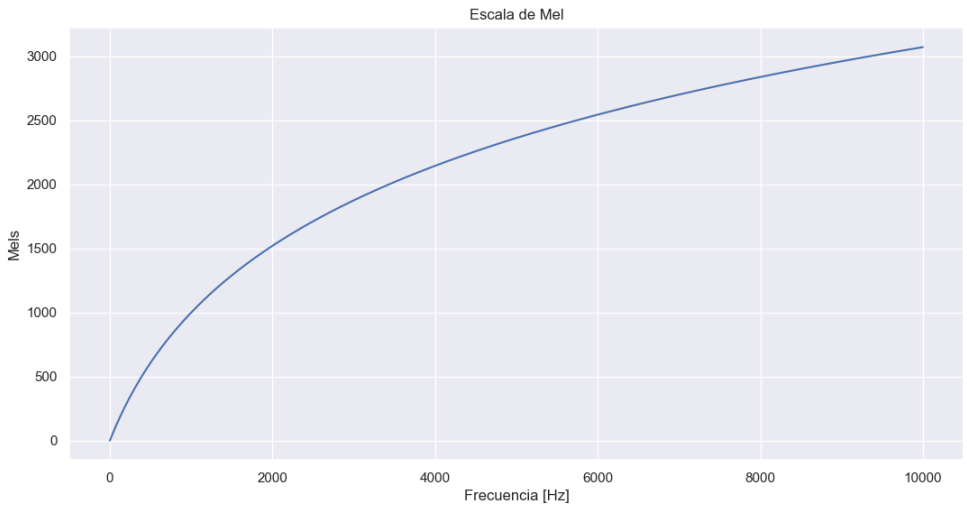


Figura 2.10: Representación de la escala de Mel

Volviendo a los mismos ejemplo de antes, ahora **A4** estaría en 549.639 mels, **B4** en 601.684 mels y **C5#** en 657.385 mels, lo que daría unas diferencias de 52.045 mels entre **B4** y **A4** y de 55.701 entre **C5#** y **B4**, valores mucho más cercanos aunque no exactamente iguales debido al carácter subjetivo de la escala de mel.

2.5.2. Bancos de filtros de Mel

La cóclea [21] es una estructura en forma de tubo enrollado situado en el oído interno. En su interior se encuentra el órgano de Corti, que contiene una serie de fibras, más largas o más cortas, para detectar diferentes frecuencias llamadas células ciliadas. La vibración de estas fibras es detectada por el órgano de Corti, que mandará la información al cerebro sobre frecuencia y amplitud dependiendo de qué fibras se muevan y cuánto.

Los bancos de filtros tratan de replicar el comportamiento en frecuencia de este órgano. Éstos son en realidad una serie de bancos de frecuencia solapados entre ellos, cada vez más estrechos a medida que aumentan en frecuencia.

Para diseñar los bancos de filtros, en primer lugar se seleccionan las frecuencias mínima y máxima que se consideran de utilidad. Éstas se seleccionan según la naturaleza de la señal, si se trabaja con voz, como es el caso, se escogerá este rango de modo que contenga el tono y los posibles armónicos más importantes de la voz humana (entre 300 Hz y 8 KHz, típicamente). Además, la frecuencia máxima deberá ser siempre menor o igual a la frecuencia de *Nyquist* ($f_{Nyquist} = \frac{f_{muestreo}}{2}$), ya que no pueden existir frecuencias por encima de ésta en la señal muestreada.

Como ejemplo, supongamos que las frecuencias seleccionadas como mínima y máxima son de 0 a 8 Khz. En primer lugar, se transforman estas frecuencias a la escala de mel usando la ecuación 2.4. Se obtiene, por tanto que los bancos de filtros estarán entre 0 y 2840 mels.

Después se divide el espacio (en mels) a partes iguales entre el número de bancos de filtros que se desee. Como la escala de mel distingue mejor las bajas frecuencias que las altas (esto es, en la Figura 2.10 las frecuencias más bajas tienen una mayor pendiente que las altas, lo que equivale a una mayor resolución en frecuencia para las bajas frecuencias), esto hará que los bloques en hercios sean más anchos a altas frecuencias.

Siguiendo con el ejemplo y suponiendo que se quieren obtener 8 bancos de filtros, se divide el espectro en la escala de mel en 8 filtros.

$$(2840 - 0)/(8 + 1) = 355$$

Cada filtro, por tanto ocupará 355 mels. Podemos definir, entonces las fronteras de estos filtros en: 0, 315.5, 631, 946.5, 1262, 1577.5, 1893, 2208.5, 2524 y 2840 mels. Finalmente transformando de nuevo estos mels a hercios, obtenemos: 0, 226.1, 525.3, 921.2, 1444.9, 2137.9, 3054.7, 4267.7, 5872.6 y 8000 hercios.

Para cada una de estas fronteras el filtro se define como:

$$H_n(k) = \begin{cases} 0, & si \quad k < f(n-1) \\ \frac{f - f(n-1)}{f(n) - f(n-1)}, & f(n-1) \leq k < f(n) \\ 1, & si \quad k = f(n) \\ \frac{f(n+1) - k}{f(n+1) - f(n)}, & f(n) < k \leq f(n+1) \\ 0, & si \quad k > f(n+1), \end{cases} \tag{2.6}$$

donde k es cada punto del espectro en potencia, n es el número del filtro y $f()$ representa las fronteras de cada banco de frecuencias definidos anteriormente.

La Figura 2.11 muestra un ejemplo de lo que sería un banco de 8 filtros para frecuencias entre 0 y 8 KHz calculadas en el ejemplo anterior. Se observa que los bancos de filtros son más estrechos a bajas frecuencias y se van ensanchando conforme se aumentan en frecuencia. Esto indica que discriminan mejor las bajas frecuencias que las altas.

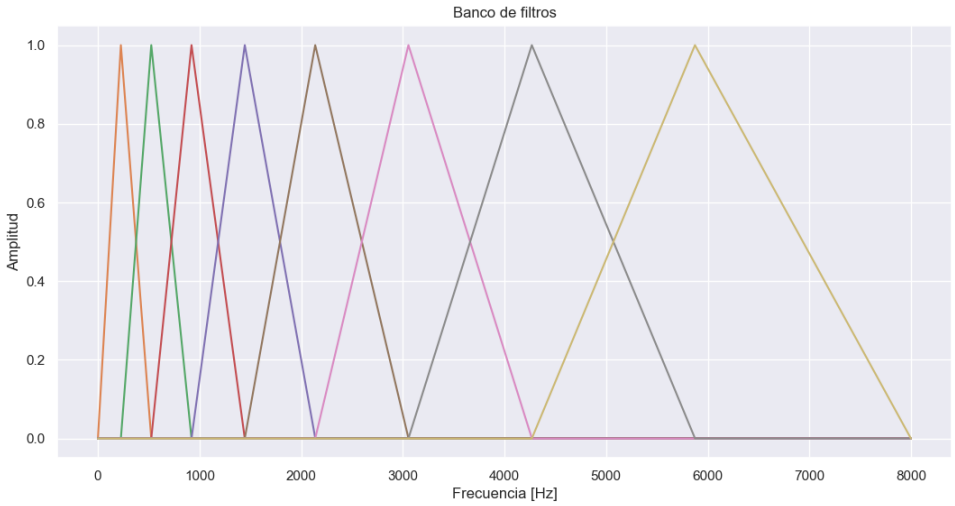


Figura 2.11: Representación de ejemplo para 8 bancos de filtros, desde 0 a 8KHz. Cada color representa un banco distinto. Se aprecia como los bancos son más discriminativos para bajas frecuencias, dado que la anchura del filtro es menor.

El siguiente paso será realizar la FFT de la señal para obtener la energía en cada banda de frecuencias. El tamaño de la ventana de la FFT determinará la resolución que tendremos en frecuencia. Hay que tener en cuenta que al tener una resolución limitada en frecuencia, los bancos de filtros no estarán definidos en las frecuencias exactas calculadas, sino en los *bins* de la FFT en los que caigan esas frecuencias.

Por último, una vez calculado el espectro en potencia $P = \frac{|FFT(x)|^2}{N}$, donde N es la longitud de la FFT usada y x la señal, se aplica la ecuación 2.6 para obtener las características finales.

2.5.3. Coecientes Cepstrales en las Frecuencias de Mel (*Mel Frecuency Cepstral Coefficients*)

Una vez calculados los bancos de filtros se calcula el logaritmo, y a continuación se aplica la Transformada Discreta del Coseno (DCT) [22]. Al igual que la transformada de Fourier que expresa cualquier señal en función de exponenciales complejas, la DCT hace lo mismo en función de cosenos. La DCT también es ampliamente utilizada en otros campos, por ejemplo para comprimir información en imágenes [23].

De este modo, la DCT comprime los datos, separándolos por frecuencias. Los componentes más bajos tendrán las frecuencias más bajas, y son las más relevantes para extraer información de este tipo de señales. Normalmente sólo se suelen coger los 13 primeros componentes, sustituyendo en ocasiones el primero por la energía de la ventana.

La Figura 2.12 muestra el esquema completo del cálculo de las MFCCs. Este esquema ilustra los distintos pasos explicados en esta sección. El primer paso es la segmentación y el enventanado, en éste, se elige un tamaño de ventana, un *offset* y una función de ventana. Después se aplica la FFT escogiendo para ello el tamaño de ventana de la FFT. Como ya se ha dicho, el tamaño de la ventana de la FFT determinará la resolución en frecuencia. Esta resolución no es demasiado crítica ya que al aplicar después los bancos de filtros, estas frecuencias serán agrupadas por bloques. En cuanto a estos bancos de filtros, hay que seleccionar el número adecuado de ellos. Finalmente se aplica el logaritmo a estos bancos y la DCT. De esta DCT se seleccionan los N primeros bancos (sustituyendo normalmente el primero por la energía), donde N es el número

escogido de componentes. Se suelen descartar los últimos porque las componentes de más alta frecuencia no suelen ser relevantes en *speech recognition*, ya que las características fundamentales para esta tarea se encuentran en las componentes de más baja frecuencia.

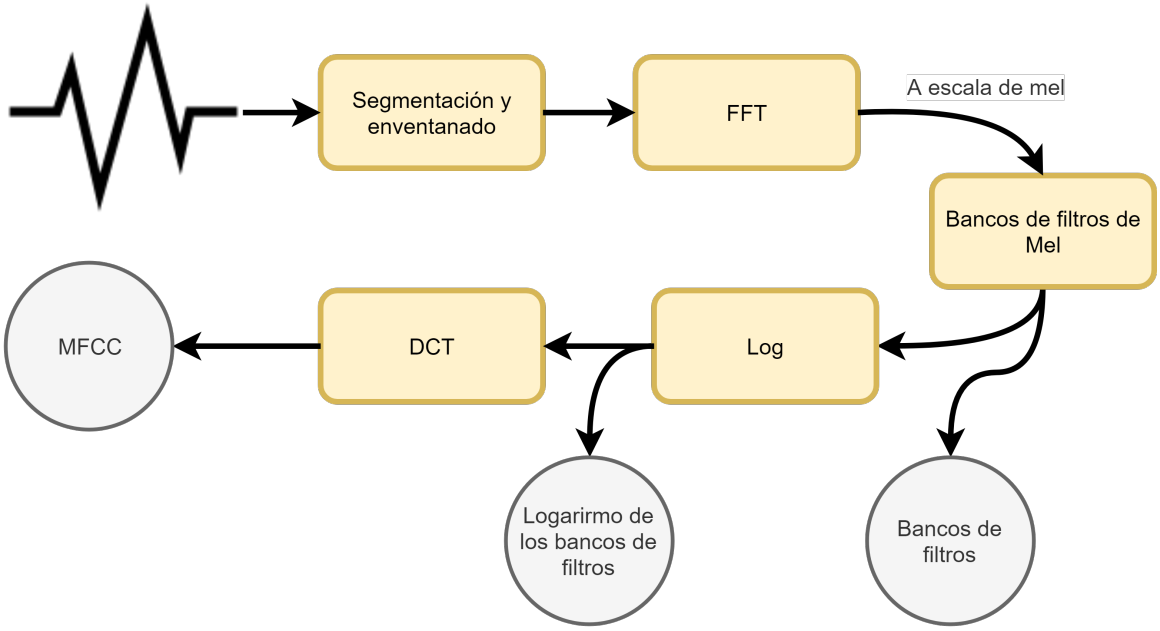


Figura 2.12: Diagrama de extracción de características con MFCC. La entrada es la señal de audio, los círculos grises representan los distintos pasos de extracción de características.

2.6. Extracción de características con aprendizaje no supervisado

También es posible aprender características discriminantes a partir de las señales, ésta es la idea detrás de los algoritmos de aprendizaje no supervisado. En teoría, la ventaja de este método es que podría adaptarse a señales de distinta naturaleza con un mismo algoritmo, por contra, exige un tiempo de entrenamiento y un esfuerzo mayor, ya que es necesario entrenar modelos, además de la elección de hiperparámetros que ofrezcan un buen resultado.

Estos algoritmos deben ser capaces de encontrar por si solos las características que definen a cada señal de un modo discriminativo.

2.6.1. Máquinas de Boltzmann

Las máquinas de Boltzmann (*Boltzmann Machines* en inglés o BM) son un tipo de red neuronal estocástica, generativa, basada en energía y típicamente binaria. Aparecieron por primera vez en el campo de *computer science* en 1983 [24], después se diseñaron otras arquitecturas para hacerlas más eficientes, como Restricted Boltzmann Machine (RBM) [5] y otros algoritmos como el Contrastive Divergence (CD) [25] que se explicará más adelante. Quizás su momento de mayor popularidad se dio al aparecer como parte del algoritmo ganador del Netflix Grand Prize [26] para su sistema de recomendaciones (*collaborative filtering*), premiado con un millón de dólares.

A parte de *collaborative filtering*, las RBM pueden ser de utilidad en tareas como: reducción de dimensionalidad, clasificación, pre-entrenamiento de redes neuronales o aprendizaje de características, entre otras.

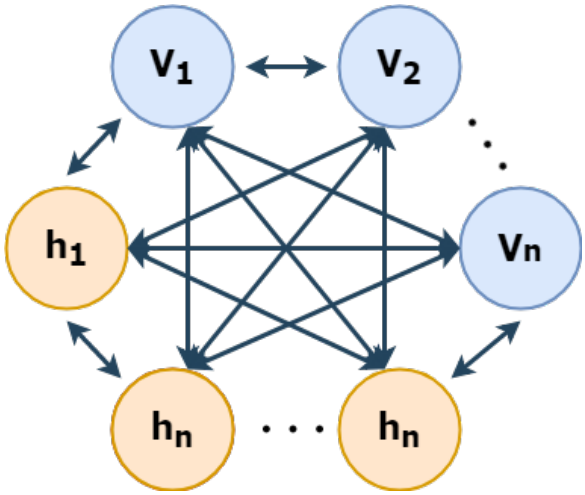


Figura 2.13: Representación gráfica de una Máquina de Boltzmann. En azul las unidades visibles, en naranja las ocultas. En un BM convencional, todas las unidades están conectadas con todas. Las flechas bidireccionales simbolizan que la información fluye en todas direcciones, es decir, la salida de cada neurona depende del resto.

Las BM se componen de unidades visibles y unidades ocultas (y las conexiones entre ellas). Las unidades visibles son las unidades por donde entrará la información al modelo (las ventanas de audio en nuestro caso). Las ocultas se encargarán de aprender correlaciones entre las unidades (ver Figura 2.13) mediante los pesos y *bias* que las conectan con el resto de unidades.

Como se ha mencionado antes, las BM pertenecen al grupo de los modelos basados en energía. Estos modelos se llaman así porque, análogamente a la mecánica estadística, la probabilidad de observar un estado de unidades visibles (v) y unidades ocultas (*hidden*) (h) depende de la

energía que tenga este estado (de hecho, esta ecuación viene de la distribución de Boltzmann en mecánica estadística). Cuanto mayor sea la energía para un estado, menor será la probabilidad de observación. Las unidades de la red suelen ser típicamente binarias, es decir, tienen dos posibles estados, 0 o 1, aunque también se pueden usar otro tipo de activaciones como las gaussianas para señales cuya naturaleza no pueda ser bien representada con estados binarios.

El estado de la red lo definimos por el estado de las neuronas visibles y ocultas (activadas o no). Concretamente, la probabilidad de que la red se encuentre en un estado determinado viene dado por:

$$P(s) = e^{-E(s)}/Z, \tag{2.7}$$

donde s representa el estado, E la función de energía y Z es la función de partición, que se define como el sumatorio de todos los posibles estados de energía para el sistema:

$$Z = \sum_{s} e^{-E(s)}. \tag{2.8}$$

Para una máquina de Boltzmann convencional, la energía de un determinado estado s se define como: $E(s) = -(\sum_{i<j} s_i s_j w_{ij} + \sum_i s_i c_i)$ donde s_i define el estado de la unidad i (normalmente definido entre 0 y 1, aunque en ocasiones entre -1 y 1), ya sea visible u oculta, w_{ij} es el peso que une la unidad i con la j (para las BM estos pesos son simétricos, $w_{ij} = w_{ji}$) y c_i define el *bias* asociado a la unidad i.

Esta arquitectura, sin embargo, no es de mucha utilidad debido a que muchas de las operaciones que se requieren para el entrenamiento son de gran complejidad [27] y en la práctica apenas se usa. Para solventar este problema, Smolensky inventó un nuevo tipo de arquitectura, conocida como Restricted Boltzmann Machine (RBM) [28] aunque él se refirió inicialmente a ella como *Harmonium*. Como consecuencia, esta última es la estructura que se usa hoy en día para trabajar con máquinas de Boltzmann.

2.6.2. Restricted Boltzmann Machine

Las RBM siguen los mismos principios que las BM, pero con una diferencia fundamental. No existen conexiones entre unidades de una misma capa (Figura 2.14), este simple detalle que no afecta demasiado a la capacidad de la red, permite realizar algunas simplificaciones importantes, reduciendo el coste computacional del entrenamiento.

Para esta arquitectura, la energía del sistema se define como:

$$E(v,h) = -\sum_{i \in visible} a_i v_i - \sum_{j \in hidden} b_j h_j - \sum_{i \in visible, j \in hidden} v_i h_j w_{ij}, \tag{2.9}$$

donde, de nuevo, v y h representan las unidades visibles y ocultas del sistema, a_i y b_j las unidades de *bias* para cada neurona visible y oculta respectivamente, y w_{ij} los pesos entre las unidades v_i y h_j.

La probabilidad de que la red se encuentre en cada estado se definirá ahora en función de v y h, tal que:

$$P(v,h) = e^{-E(v,h)}/Z. \tag{2.10}$$

Esta estructura permite que el cálculo de una capa solo dependa de la capa contraria. Así, para calcular los actuales estados ocultos (las unidades h), solo se necesitan las unidades visibles y viceversa. Estas probabilidades de activación condicionadas se calculan como:

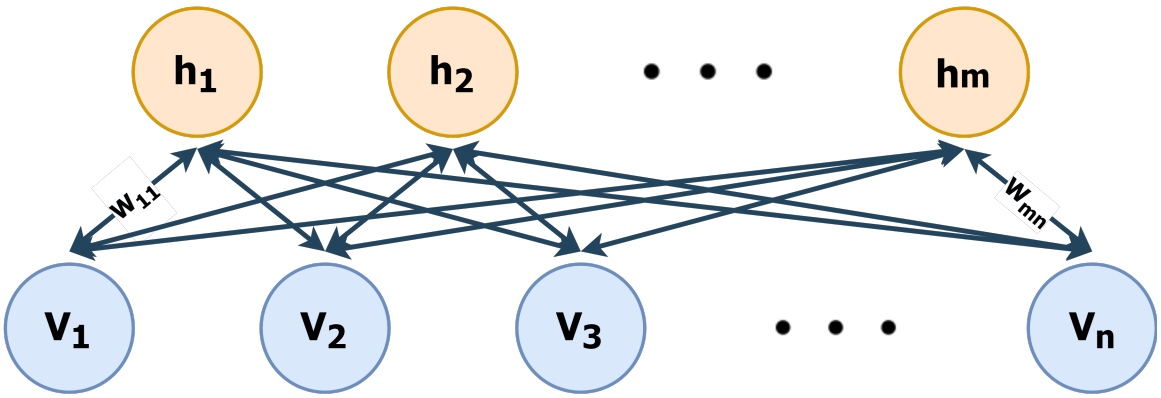


Figura 2.14: Representación gráfica de una RBM. En azul las unidades visibles, en naranja las ocultas. No existen conexiones entre unidades de la misma capa.

$$P(h_i|v) = \frac{1}{1 + e^{-b_i - vw_i}} = \sigma(b_i + vw_i), \tag{2.11}$$

y de forma similar para las visibles como:

$$P(v_j|h) = \frac{1}{1 + e^{-a_j - w_j h}} = \sigma(a_j + hw_j), \tag{2.12}$$

donde σ es la función sigmoide definida en la ecuación 2.13.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.13}$$

Una vez calculadas estas probabilidades, el muestreo de las activaciones consisten en poner cada una de estas unidades a 1 si esta probabilidad es mayor que un umbral, definido por una distribución uniforme entre 0 y 1. Esto se conoce como muestreo.

Entrenamiento

Para el entrenamiento de las RBM se utiliza típicamente el **descenso por gradiente estocástico** (o ascenso por gradiente si se emplea el negativo de los gradientes). Estos gradientes son el vector del error calculado para la red en función de los parámetros de esta. Al descender por gradiente, estos parámetros van cambiando para tratar de llegar a un mínimo más óptimo. El término estocástico hace referencia al hecho de que no se emplee el conjunto total de los datos de entrenamiento para cada iteración, si no pequeños subconjuntos o *batches*. Esto se hace así porque de otro modo produciría gradientes más pequeños para los pesos según Hinton (ver [29] punto 4).

Estos gradientes se calculan sobre $P(v)$, dado que los datos de entrenamiento son sólo las neuronas visibles. Esta $P(v)$ se define como:

$$P(v) = \sum_h e^{-E(v,h)}/Z,$$

y los gradientes para las RBM como:

$$\frac{\partial log(P(v))}{\partial w_{ij}} = \langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model} \tag{2.14}$$

$$\frac{\partial log(P(v))}{\partial a_i} = \langle v_i \rangle_{data} - \langle v_i \rangle_{model} \tag{2.15}$$

$$\frac{\partial log(P(v))}{\partial b_i} = \langle h_i \rangle_{data} - \langle h_i \rangle_{model} \tag{2.16}$$

donde $P(v)$ es la probabilidad de observación de las neuronas visibles v, *data* es la distribución de los datos de entrenamiento y *model* la distribución interna del modelo. El operador $\langle x \rangle$ define la media de x.

La distribución de los datos de entrenamiento se calcula a partir de las unidades visibles mapeadas a estos datos de entrenamiento y de las unidades ocultas que resultan del muestreo de estos datos. De tal modo que si nuestros datos de entrenamiento son v_1, y las activaciones de las unidades ocultas calculadas a partir de estos datos son h_1, la distribución $\langle v_1 h_1 \rangle$ será la multiplicación matricial de v_1 por h_1.

Sin embargo, para la distribución del modelo esto no es tan sencillo. En teoría, estos datos han de salir de un estado de equilibrio del modelo, es decir muestrear las unidades visibles a partir de las ocultas y las ocultas a partir de las visibles hasta que no se aprecien grandes cambios de una iteración a otra. Esto se conoce como *muestreo de Gibbs* [30]. La Figura 2.15 muestra este proceso.

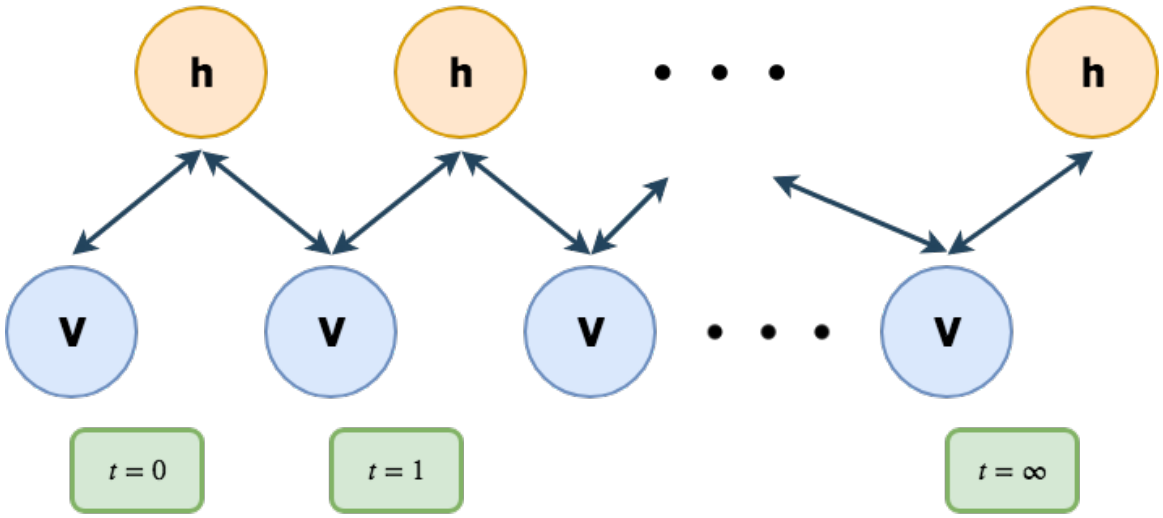


Figura 2.15: Muestreo de Gibbs para una RBM. Se muestrean las unidades ocultas a partir de las visibles y las visibles a partir de las ocultas hasta alcanzar el equilibrio.

Como se puede intuir, este proceso es costoso porque el número de iteraciones necesarias para que se alcance el estado de equilibrio puede ser grande. Afortunadamente, existe una aproximación a este proceso conocido como Divergencia Contrastiva (*Contrastive Divergence* en inglés o CD) [25]. Ésta consiste en obtener esta muestra (unidades visibles y ocultas) partiendo de las unidades ocultas obtenidas desde la muestra de entrenamiento, como se ilustra en la Figura 2.16. En realidad, este proceso se puede repetir k veces (lo que se conoce como CD-k) pero lo más común es hacerlo sólo una vez (es decir, aplicar CD-1) ya que suele ofrecer buenos resultados [31]. Esto nos indica que no es necesario alcanzar el estado de equilibrio para realizar el aprendizaje del modelo.

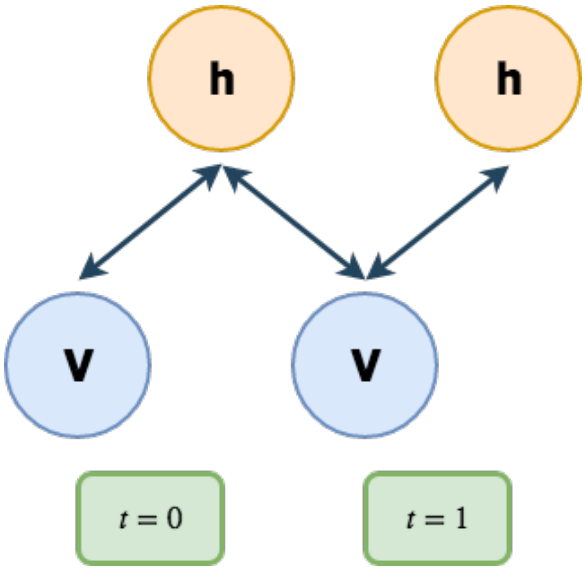


Figura 2.16: Ejemplo de Contrastive Divergence para k=1. Similar al muestreo de Gibbs, pero realizando sólo k iteraciones. Este algoritmo reduce drásticamente el coste computacional de entrenar una RBM.

Distintos tipos de neuronas en RBM

Aunque inicialmente el concepto de las RBM parte de unidades binarias, existen otros tipos de neuronas que se pueden usar y que ofrecen mayor potencial porque pueden adaptarse a distintos tipos de datos.

Un tipo de unidad muy utilizada en las RBM, mayormente para las unidades visibles, son las unidades gaussianas. El muestreo de las activaciones de estas unidades son muestras de una distribución gaussiana. Esta distribución gaussiana se define usando el campo medio de la neurona ($wh+a$ en el caso de las visibles) como media dejando la desviación estándar a 1, ya que se suele trabajar con datos normalizados (aunque también es posible aprender esta desviación estándar a partir de los datos [32][33]). Este tipo de unidad viene bien para caracterizar muchos tipos de datos porque, a diferencia de las binarias, su salida puede adaptarse a cualquier tipo de valores. Si se trabaja con sonido, por ejemplo, las muestras de audio pueden aproximarse bien con una distribución gaussiana.

La energía para este tipo de RBM, suponiendo que las contrarias (unidades ocultas) sean binarias o alguna variante de éstas se define típicamente como: $E(v,h) = -v^T W h + \frac{1}{\sigma^2}||v - b||^2 - h^T a$, donde W es la matriz de pesos, σ es la desviación típica, v las unidades visibles, h las unidades ocultas y a y b son las unidades de *bias* para las unidades ocultas y visibles, respectivamente.

Otro tipo de unidades usadas son las unidades ReLU (*Rectified Linear Unit*). Éstas se definen como $ReLU(x) = max(0, x)$. Cada una de estas unidades puede interpretarse como un bloque de infinitas unidades binarias. Esto es, si se sustituyesen cada una de las neuronas de una capa de neuronas binarias por infinitas neuronas binarias, pero cada una de éstas restándole un offset progresivamente decreciente (-0.5, -1.5, -2.5, ...), las activaciones de estas neuronas en su conjunto seguirían una distribución *softplus*:

$$\sum_{i=1}^{N} \sigma(x - i + 0{,}5) \approx log(1 + e^x),$$

donde x es la unidad en cuestión, σ es la función sigmoide y N el número de neuronas binarias

con las que se quiera aproximar esta función *softplus*.

Estas unidades se conocen como Steped Sigmoid Units (SSU) [32] y pueden aproximarse bien con una unidad ReLU como se muestra en la Figura 2.17. Al estar estas unidades basadas en unidades binarias, la definición de energía no cambia, seguiría siendo la definida en la ecuación 2.9. Por último, como todas las unidades de la RBM son ruidosas, ya que el ruido actúa como regularizador, las activaciones en realidad usadas son gaussianas con media $ReLU(x)$ y desviación estándar $\sigma(x)$, donde x es el campo medio y sigma la función sigmoide.

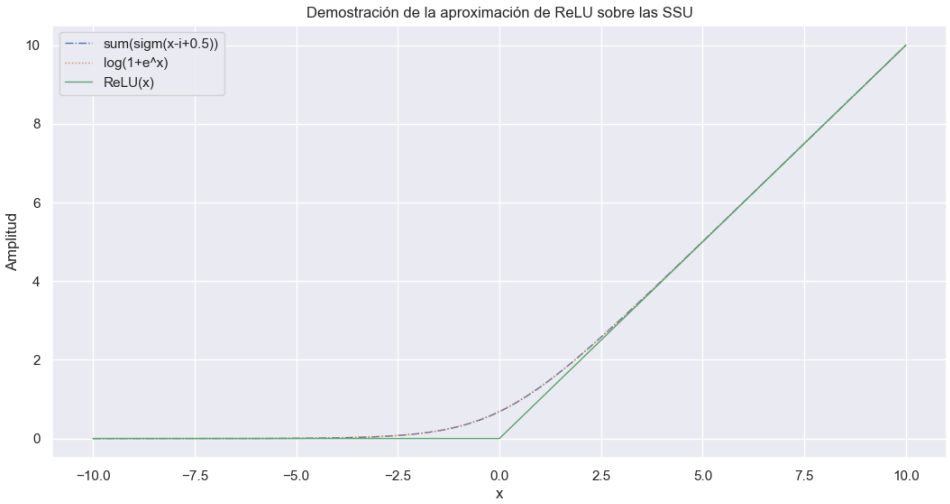


Figura 2.17: Aproximación de las SSU por una ReLU

2.7. Fundamentos de los Modelos Ocultos de Markov

Los modelos ocultos de Markov(HMM), son modelos de probabilidad basados en las cadenas de Markov de las que no se conoce el estado en el que se encuentra el modelo en cada momento, sólo se conocen las emisiones (u observaciones) que puedan salir de cada estado. Estas observaciones son los parámetros observables de cada estado.

Estos modelos, en alguna de sus variantes, son muy utilizados en aplicaciones de procesado de lenguaje natural (*Natural Language Processing* en inglés o NLP) como el etiquetado gramatical y en algunas otras áreas del aprendizaje máquina como el reconocimiento de patrones o en bioinformática como la predicción de la expresión genética. Por eso, aunque este trabajo se centrará en la clasificación con redes neuronales recurrentes (explicadas en la sección 2.8), merece la pena explicar estos modelos.

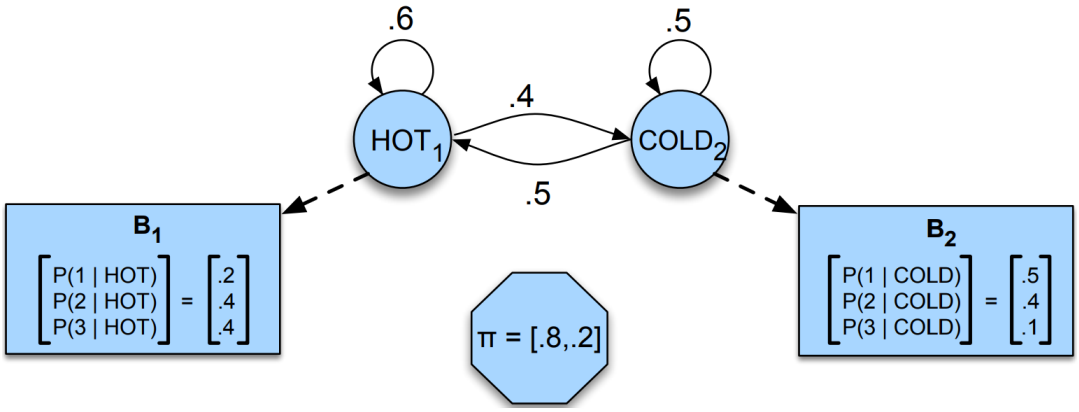


Figura 2.18: Modelo Oculto de Markov con dos estados (HOT y COLD), y sus probabilidades de emisión (B1 y B2) para las posibles observaciones (1, 2 y 3). Imagen extraída de [7].

La Figura 2.18 muestra la representación gráfica de un HMM con dos estados (HOT y COLD). Para cada instante en el que el modelo se encuentre en cada uno de estos estados, se emitirá una observación (1, 2 o 3) según la probabilidad de emisión asociada a cada estado (B1 y B2).

Para el ejemplo de la Figura 2.18, la matriz A, de probabilidades de transición de un estado a otro, se definiría como:

$$A = \begin{pmatrix} P(q_{t+1} = HOT|q_t = HOT) & P(q_{t+1} = COLD|q_t = HOT) \\ P(q_{t+1} = COLD|q_t = COLD) & P(q_{t+1} = HOT|q_t = COLD) \end{pmatrix} = \begin{pmatrix} 0{,}6 & 0{,}4 \\ 0{,}5 & 0{,}5 \end{pmatrix} \quad (2.17)$$

la matriz B, de emisiones de cada estado, como:

$$\begin{aligned} B &= \begin{pmatrix} P(o_t = 1|q_t = HOT) & P(o_t = 12|q_t = HOT) & P(o_t = 13|q_t = HOT) \\ P(o_t = 11|q_t = COLD) & P(o_t = 12|q_t = COLD) & P(o_t = 13|q_t = COLD) \end{pmatrix} \\ &= \begin{pmatrix} 0{,}2 & 0{,}4 & 0{,}4 \\ 0{,}5 & 0{,}4 & 0{,}1 \end{pmatrix} \end{aligned} \quad (2.18)$$

y la matriz Π, o matriz de entrada, como:

$$\Pi_F = \begin{pmatrix} P(q_0 = HOT) & P(q_0 = COLD) \end{pmatrix} = \begin{pmatrix} 0{,}8 & 0{,}2 \end{pmatrix} \quad (2.19)$$

Al ser estas matrices probabilidades conjuntas, se debe cumplir para la matriz A que:

$$\sum_{j}^{N} a_{ij} = 1,$$

siendo a_{ij} la probabilidad de pasar del estado i al j y lo mismo para el resto de matrices. Es decir, estando en el estado q_i, la probabilidad de quedarse o irse a cualquier otro estado ha de sumar 1.

De igual modo, para la matriz B:

$$\sum_{o}^{O} b_i(o) = 1 \quad \forall i,$$

donde, en este caso, O son todas las posibles observaciones (1, 2 ó 3 en el ejemplo) y $b_i(o)$ representa la probabilidad $P(o|q = i)$. Es decir, cada estado debe emitir siempre una observación (con excepción de los estados silenciosos, en los que no se emite ninguna observación, como pueden ser el estado inicial o el final).

En ocasiones, cuando no todos los estados tienen la misma probabilidad de salida, también puede ser útil tener un estado de salida, que se modelará con:

$$\Pi_F = P(q_T|q_{T-1}),$$

donde T es el instante final de las observaciones.

2.7.1. Cadenas de Markov

Una cadena de Markov consiste en una serie de estados interconectados entre ellos, similar a una máquina de estados, pero con la peculiaridad de que el cambio de un estado a otro es no-determinista, es decir, depende de una probabilidad.

El número de estados pasados que influyan en la transición del estado actual al siguiente determinará el orden de la cadena de Markov. Así, si el siguiente estado solo depende del actual será una cadena de primer orden, si depende del actual y el anterior será de segundo orden etc...

Dicho matemáticamente, para una cadena de Markov de primer orden:

$$P(q_{t+1}|q_t, q_{t-1}...q_0) = P(q_{t+1}|q_t),$$

siendo q_t el estado en el instante t actual.

2.7.2. Modelos Ocultos de Markov

Normalmente para los modelos ocultos de Markov se utilizan cadenas de Markov de primer orden, aunque existen también modelos con cadenas de orden superior [34].

Además los HMM asumen independencia de la salida, esto es, la emisión en cada instante t sólo depende del estado en ese instante, es decir:

$$P(o_t|q_0...q_t) = P(o_t|q_t),$$

siendo o_t la emisión en el instante actual t y q_t el estado actual. En este trabajo se explicarán los modelos típicos (con cadenas de primer orden).

Como se ha visto para el ejemplo de la Figura 2.18, un HMM puede definirse por un conjunto de matrices $\lambda = (A, B, \Pi)$. La matriz A representa la probabilidad de transición de un estado a otro, la matriz B la probabilidad de las observaciones de cada estado y la matriz Π la probabilidad de estado inicial. Esto es:

$$A = P(q_{t+1}|q_t)$$

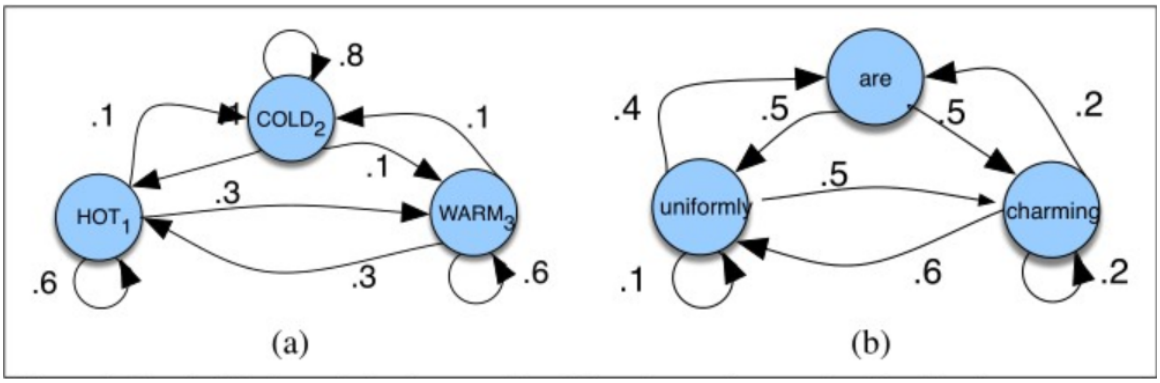


Figura 2.19: Dos ejemplos de cadenas de Markov de primer orden con tres posibles estados cada una (HOT, COLD y WARM en la izquierda; *uniform*, *charming* y *are* en la derecha). Los números sobre cada flecha indican las probabilidades de transición de un estado a otro. Imagen extraída de [7].

$$B = P(O|q_t)$$
$$\Pi = P(q_1|q_0),$$

siendo O la lista de todas las observaciones para todos los instantes t y Q la de los estados. El número de estados suele definirse como N.

Para el caso más simple, todos los estados podrán ser accesibles desde cualquier otro estado en un solo paso, esto se conoce como modelo *ergódico*. Sin embargo, para trabajar con secuencias esto no es lo más adecuado y se suele usar otro tipo de modelo conocido como modelo de Bakis o modelo de izquierda a derecha (ver Figura 2.20), en el que los estados sólo pueden alcanzar su mismo estado o los Δ siguientes estados superiores.

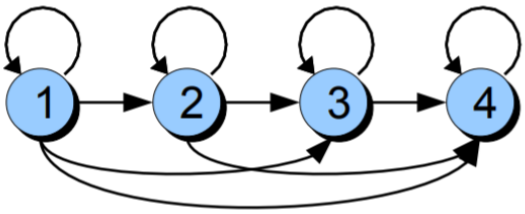


Figura 2.20: Modelo de Bakis [7]. Los diferentes estados sólo se conectan consigo mismo y con Δ estados superiores, siendo $\Delta = 3$ en este caso.

Existen tres problemas de interés [3] para un HMM. Estos problemas permiten calcular las distribuciones del modelo con respecto a los datos de entrenamiento, que permitirán aprender y clasificar secuencias. En definitiva, calcular estas distribuciones es lo que permite hacer del HMM un clasificador. Estos problemas son:

- Estimación de la verosimilitud: Probabilidad de observar una determinada secuencia de observaciones ($P(O|\lambda)$). Veremos que hay, principalmente, dos metodologías para estimar esta verosimilitud. Una de ellas es mediante el uso de un algoritmo de flujo hacia adelante o *Forward* [7] en el que desde la posición inicial se va avanzando y para el cálculo de la verosimilitud se tiene en cuenta el estado y las observaciones del tiempo anterior. La otra metodología es partir del estado final y realizar una valoración retrospectiva, algoritmo *Backward* [7].

- Decodificación: Dada una secuencia de observaciones, estimar cuáles fueron los estados que con mayor probabilidad la produjeron ($P(Q|O)$).
- Aprendizaje: Dada una serie de secuencias de observaciones, estimar los parámetros que mejor se ajustan a esta serie.

Estimación de la verosimilitud. Enfoques *forward* y *backward*

La estimación de la verosimilitud ($P(O)$) no se usará directamente para entrenamiento ni decodificación de fonemas, aunque se usa internamente dentro del entrenamiento del modelo. Para calcularla se utiliza el algoritmo *forward*.

Éste consiste en ir calculando para cada instante de tiempo t y para cada estado la probabilidad de observación o_t teniendo en cuenta las probabilidades acumuladas en el instante anterior, en lo que se conoce como un diagrama de Trellis (ver Figura 2.21).

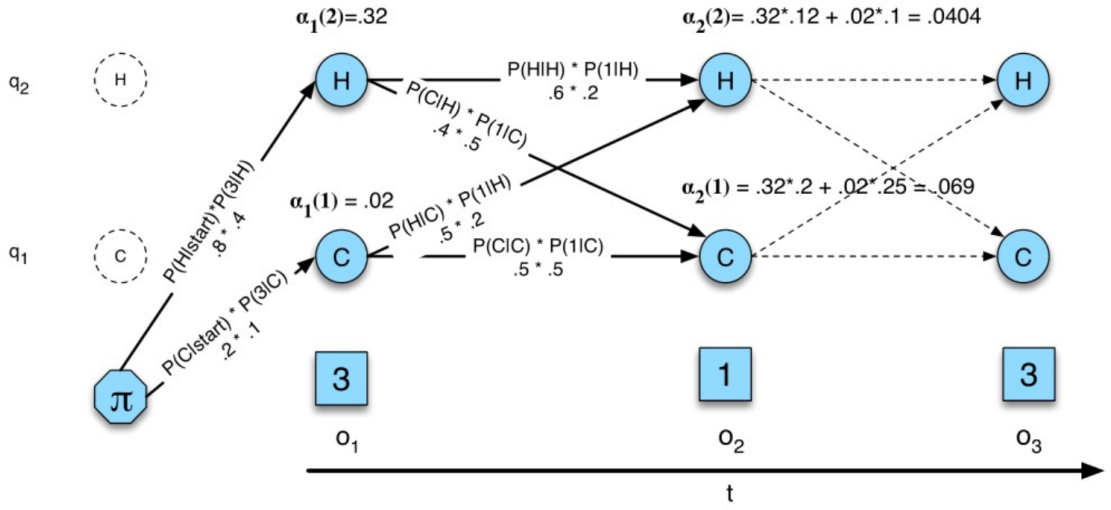


Figura 2.21: Algoritmo *forward*. Cálculo del algoritmo *forward*, para las matrices definidas en las ecuaciones 2.17, 2.18 y 2.19 y suponiendo que las observaciones son 3, 1, 3. Imagen extraída de [7].

Para el ejemplo de la Figura 2.21, la probabilidad *forward* para el instante $t = 1$ y el estado 1 ($\alpha_1(1)$) se calcularía como la probabilidad de ir al estado 1 (C en este caso), por la probabilidad de que este estado emita la primera observación (3 en el ejemplo), es decir $\alpha_1(1) = \pi_1 b_1(3)$, donde π_i y $b_i(j)$ son elementos de las matrices Π y B que definen al HMM. Se realizaría este proceso para todos los estados en este primer instante t. Seguidamente, para el instante $t = 2$, se calcularía la probabilidad del mismo modo, pero teniendo en cuenta todos los posibles estados anteriores. Es decir, la probabilidad $\alpha_2(1)$ (probabilidad *forward* para el instante 2 y el estado 1) se calcularía como la $\alpha_2(1) = \alpha_1(1)a_{11}b_1(1)+\alpha_1(2)a_{21}b_1(1)$, esto es, la probabilidad de que viendo de cada posible estado anterior, se de la siguiente observación. De nuevo este paso se realiza para todos los posibles estados. Este proceso continua hasta que no queden más observaciones.

El procedimiento sería por tanto como se muestra en *Algoritmo* 1, donde $\alpha_t(i)$ es el valor calculado del algoritmo *forward* para el estado i en el instante t, π_j es la probabilidad de que el estado inicial sea j (definido en la matriz Π), $b_i(o_t)$ es la probabilidad de que el estado i emita la observación o_t (definido en la matriz B), a_{ij} es la probabilidad de transición del estado i al estado j (definido en la matriz A), N el número total de estados y T el número total de observaciones..

Algoritmo 1 Algoritmo *forward*

Inicialización:

$$\alpha_1(j) = \pi_j b_j(o_1) \qquad \forall 1 \leq j \leq N$$

Iteración :

$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_j(o_t) \qquad \forall 1 \leq j \leq N, \quad \forall 1 \leq t \leq T$$

Fin :

$$P(O) = \sum_{i=1}^{N} \alpha_T(i)$$

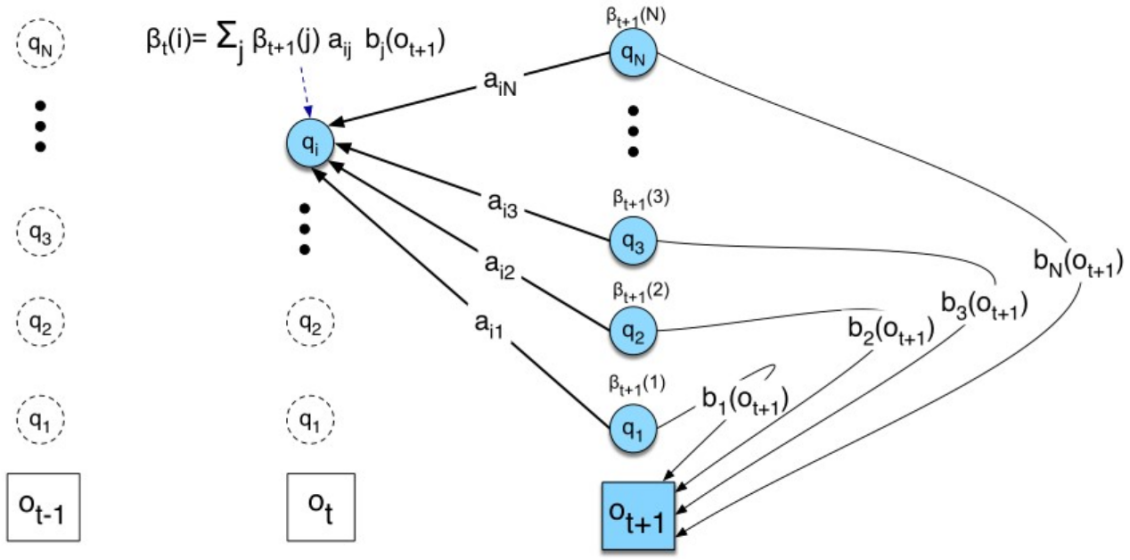


Figura 2.22: Algoritmo *backward*. Similar al algoritmo *forward*, calcula la probabilidad P(O) sobre un diagrama de Trellis pero desde la última observación. Imagen extraída de [7].

Similar al algoritmo *forward* existe el algoritmo *backward*. Este calcula la misma probabilidad de observación P(O), pero empezando desde el final de las observaciones (ver Figura 2.22). El algoritmo *backward* será de utilidad en el entrenamiento del modelo. Este procedimiento se muestra en *Algoritmo* 2, donde $\beta_t(i)$ es el valor calculado del algoritmo *backward* para el estado i en el instante t, N el número total de estados y T el número total de observaciones.

Decodificación. El algoritmo de Viterbi

La decodificación ($P(Q|O)$) calcula los estados más probables, dadas unas observaciones O. Esto se usará para extraer la secuencia de fonemas presente en la señal. Éstos se calculan mediante el algoritmo de *Viterbi*. Similar a como se hizo en el algoritmo *forward*, se desarrollan las observaciones en un modelo de Trellis, y se calculan las probabilidades de emisión en cada instante t para cada estado, pero en este caso teniendo en cuenta sólo el camino más probable anterior y desechando el resto de posibles caminos (ver Figura 2.23). Finalmente, se selecciona el camino con los estados con mayor probabilidad (ver Figura 2.24).

En el ejemplo de la Figura 2.23, el algoritmo para el primer instante de tiempo se calcularía exactamente igual que en el *forward*, es decir, $v_1(1) = \pi_1 b_1(3)$, donde π_i y $b_i(j)$ son elementos de las matrices Π y B que definen al HMM. Seguidamente, para el instante $t = 2$, se calcularía

Algoritmo 2 Algoritmo *backward*

Inicialización:

$$\beta_T(i) = 1 \qquad \forall 1 \leq i \leq N$$

Iteración :

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j) \qquad \forall 1 \leq j \leq N, \quad \forall 1 \leq t \leq T$$

Fin :

$$P(O) = \sum_{j=1}^{N} \pi_j b_j(o_1) \beta_1(j)$$

de forma similar, pero quedándonos sólo con el camino más probable. Es decir, para el instante $t = 2$ y el estado 1, $v_2(1)$ sería el que diera máxima probabilidad entre $v_1(2)$ por la probabilidad de ir al estado 1 y emitir la siguiente observación ó $v_1(1)$ y de nuevo, ir al estado 1 y emitir la siguiente observación, es decir $v_2(1) = max(v_1(2)a_{12}b_1(1), v_1(1)a_{11}b_1(1))$. Esto de nuevo, se realizaría para todos los estados y todas las observaciones. Finalmente, se calcula ct (el camino más probable) buscando para cada instante de tiempo el estado con la mayor probabilidad v_t.

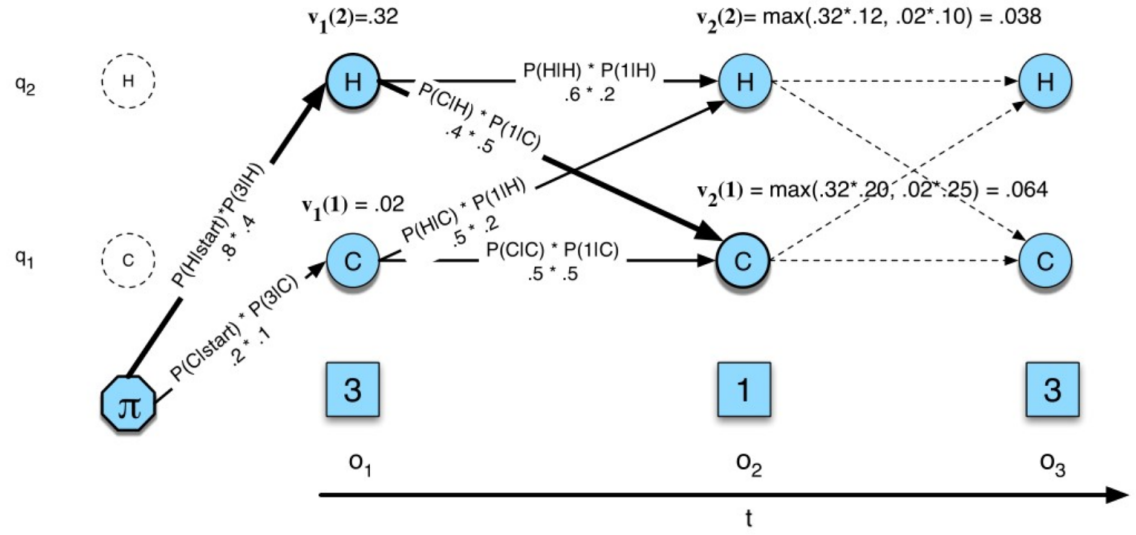


Figura 2.23: Cálculo de las probabilidades de Viterbi, para las matrices definidas en las ecuaciones 2.17, 2.18 y 2.19 y suponiendo que las observaciones son 3, 1, 3. Imagen extraída de [7].

El procedimiento se muestra en *Algoritmo* 3, donde ct contiene el camino de estados más probable dada la secuencia de observaciones O y el vector v contiene la probabilidad de dicho camino.

Aprendizaje. El algoritmo Baum-Welch

El objetivo del aprendizaje es ajustar el modelo $\lambda = (A, B, \Pi)$ a los datos de entrenamiento. Para esto, se emplea el algoritmo Baum-Welch (también conocido como *forward-backward*), que es un caso especial del algoritmo de Esperanza-Maximización (EM) [35].

El algoritmo de EM consta de dos fases que se repiten hasta alcanzar la convergencia. La fase de Esperanza (E) y la fase de Maximización (M). En la fase E, se calculan las probabilidades que definen a la matrices A (probabilidades de transición) y B (probabilidades de observación)

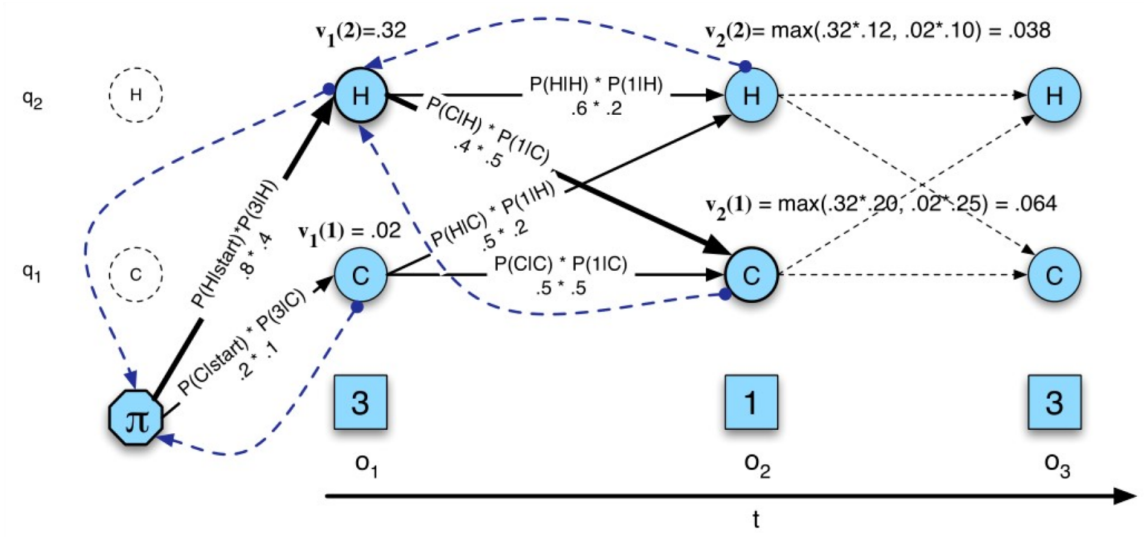


Figura 2.24: *Backtrace*. Selección de el camino de Viterbi con mayor probabilidad, para las matrices definidas en las ecuaciones 2.17, 2.18 y 2.19 y suponiendo que las observaciones son 3, 1, 3. Las flecha azul con linea discontinua muestra este camino *backtrace*. Imagen extraída de [7].

Algoritmo 3 Algoritmo de Viterbi

Inicialización:

$$v_1(j) = \pi_j b_j(o_1) \qquad \forall 1 \leq j \leq N$$

$$ct_1(j) = 0 \qquad \forall 1 \leq j \leq N$$

Iteración :

$$v_t(j) = \max_{i=1^N} v_{t-1}(i) a_{ij} b_j(ot) \qquad \forall 1 \leq j \leq N, \quad \forall 1 \leq t \leq T$$

$$ct_t(j) = \operatorname*{argmax}_{i=1}^{N} v_{t-1}(i) a_{ij} b_j(ot) \qquad \forall 1 \leq j \leq N, \quad \forall 1 \leq t \leq T$$

Fin :

$$v_T = \max_{i=0}^{N} v_T(i)$$

$$ct_T = \operatorname*{argmax}_{i=1}^{N} v_T(i)$$

dadas las observaciones de entrenamiento. Es decir, las probabilidades de cambio de estado para la matriz A se calculan como $\xi_t(i,j) = P(q_t = i, q_{t+1} = j|O)$ y las probabilidades de emisión de O para cada instante temporal para la matriz B como $\gamma_t(j) = P(q_t = j|O)$.

Después, en la fase M, se ajustan estos parámetros para que el modelo se ajuste a los datos de entrenamiento.

Este procedimiento se muestra en el *Algoritmo* 4 donde $\alpha_t(i)$ representa el valor del algoritmo *forward* para el instante t en el estado i, $\beta_t(j)$ es el valor del algoritmo *backward* para el instante t en el estado j, a_{ij} es la probabilidad de transición del estado i al j (extraído de la matriz A) y $b_j(o)$ es la probabilidad de emitir o en el estado j.

Algoritmo 4 Algoritmo de Baum-Welch

Inicialización: Se inicializan las matrices A y B

Iteración **:** Repetir hasta convergencia.

Fase E:

$$\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{\alpha_T(q_F)} \qquad \forall 1 \leq j \leq N, \quad \forall 1 \leq t \leq T$$

$$\xi_t(i,j) = \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{\alpha_T(q_F)} \qquad \forall 1 \leq i,j \leq N, \quad \forall 1 \leq t \leq T$$

Fase M:

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T} \xi_t(i,j)}{\sum_{t=1}^{T}\sum_{k=1}^{N} \xi_t(i,k)}$$

$$\hat{b}_j(v_k) = \frac{\sum_{t=1\, tal\, que\, o_t=v_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$$

2.7.3. Modelos Ocultos de Markov aplicados al reconocimiento de fonemas

Para aplicar HMM al reconocimiento de fonemas, típicamente se diseña un submodelo HMM para cada fonema. Este submodelo es una arquitectura Bakis (ver Figura 2.20) con un mínimo de tres estados, ya que estos tres estados modelan el comienzo, la parte media y el final del fonema, que normalmente son partes bastante diferenciadas.

Después, existen básicamente dos maneras de usar estos submodelos: con dependencia o independencia de contexto [36]; cuya diferencia está en si se tienen en cuenta los fonemas anterior y/o posterior en el modelo o no.

En el modelo con independencia de contexto no se tienen en cuenta los fonemas anterior y posterior. Este modelo, por tanto, no ofrece el modelado más exacto de los datos ya que en el habla normal, el paso de un fonema a otro no es automático, hay unos instantes de transición entre ellos y por tanto las características en el primer y último instante del fonema pueden diferir de forma significativa según los fonemas adyacentes a éste. En esta arquitectura, las transiciones entre estados se realizan pasando por estados silenciosos (estados que no emiten observaciones). Esta arquitectura se ilustra en la Figura 2.25, panel izquierdo, obsérvese como la transición entre fonemas siempre pasa por el estado de silencio de la derecha y desde éste existe una transición al estado silencio de la izquierda que permite transiciones a otros fonemas.

Los modelos con dependencia de contexto no tienen este problema. En éstos se modela un submodelo para cada fonema y contexto. Por ejemplo, en la frase "She had your dark suit", que se compone de los fonemas /sh/, /ix/, /hv/, /eh/, /dcl/, /jh/, /ih/, /dcl/, /d/, /ah/, /kcl/, /k/, /s/, /ux/, /q/, (ver Figura 2.3) teniendo sólo en cuenta el contexto de la derecha, se segmentaría en:

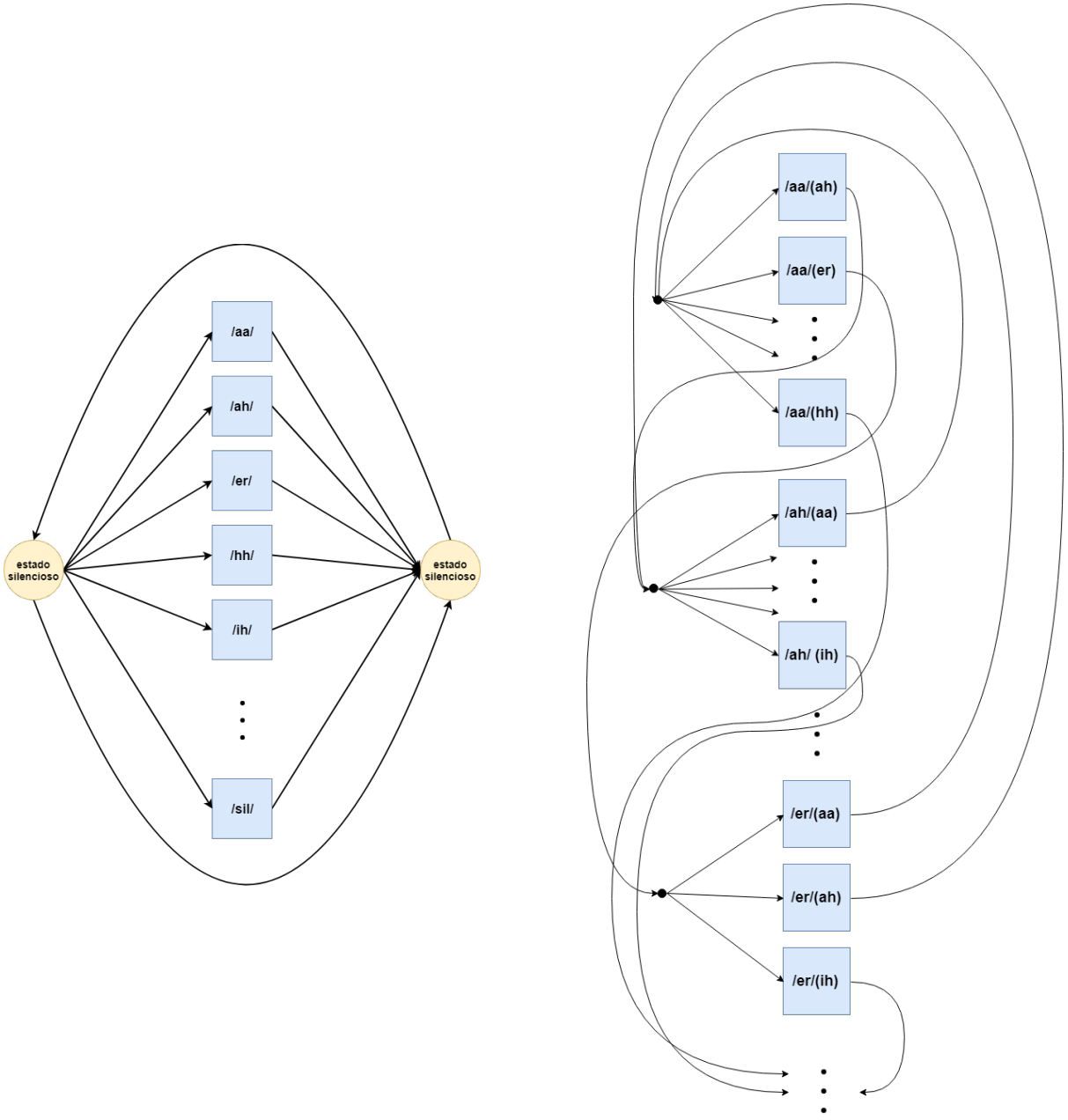


Figura 2.25: Modelos HMM para el reconocimiento de fonemas. Los cuadrados simbolizan submodelos HMM Bakis de al menos tres estados. Panel izquierdo: HMM con independencia de contexto. Panel derecho: HMM con dependencia del contexto a la derecha

/sh/(/ix/), /ix/(/hv/), /hv/(/eh/), /eh/(/dcl/), /dcl/(/jh/), /jh/(/ih/), /ih/(/dcl/), ... donde el fonema entre paréntesis representa el contexto y el fonema a su izquierda es el fonema actual. Modelar todos estos fonemas con su contexto implicaría tener M^2 submodelos (suponiendo que cualquier par de fonemas fuese posible), donde M es el número de clases de fonemas existente. Esta arquitectura, por tanto sólo será viable si hay suficientes datos para entrenar tantos estados. Esta arquitectura se ilustra en la Figura 2.25 panel derecho donde se observa que el número de posibles transiciones se ve incrementado, aumentando, por tanto, el número de datos necesarios para un correcto entrenamiento.

Por último, también hay que tener en cuenta que la matriz B (probabilidad de las observaciones de cada estado) contiene un número de observaciones totales finito (1, 2 y 3 en el ejemplo de las ecuaciones 2.17, 2.18 y 2.19), por lo que no pueden usarse como observaciones los valores directamente calculados con extractores de características como MFCC porque daría lugar a una matriz B de infinitas columnas, dado que los valores MFCC están definidos en un espacio continuo.

Para esto hay varias posibles soluciones. Se puede crear un diccionario de símbolos y mapear cada vector de características al símbolo más cercano. Esto se conoce como Cuantificación Vectorial(VQ) [37] pero no da buenos resultados porque se pierde demasiada información debido a que la aproximación de cada observación al símbolo más cercano puede ser muy inexacta.

En su lugar, es más común usar Modelos Mixtos de Gaussianas(GMM) [38] para calcular directamente la probabilidad de observación como:

$$b_j(o_t) = \sum_{m=1}^{M} c_{jm}\mathcal{N}(o_t, \mu_{jm}, \Sigma_{jm}),$$

donde c_{jm} son los coeficientes de la mezcla de M gaussianas en el estado j, y $\mathcal{N}(o_t, \mu_{jm}, \Sigma_{jm})$ es una distribución normal en o_t con media μ_{jm} y matriz de covarianza Σ_{jm} [39].

Existen por supuesto más aproximaciones, por ejemplo usando redes neuronales profundas [40] o máquinas de vector soporte (SVM) [41].

2.8. Redes Neuronales

Las redes neuronales son modelos matemáticos inspirados en las neuronas y sus conexiones en el cerebro. Estas neuronas están normalmente organizadas por capas.

Quizás su versión más simple y más conocida es una red *feed-forward*, en la que las neuronas, estructuradas por capas, tienen conexiones con las neuronas de cada capa anterior y siguiente pero no con neuronas de su misma capa.

Estas conexiones se definen como $x^{(2)} = f(w^{(1)}x^{(1)} + b^{(1)})$, donde $x^{(1)}$ es el vector fila de neuronas de la capa 1, $w^{(1)}$ son los pesos que conectan la capa 1 con la 2 y $b^{(1)}$ se corresponde con el *bias* y f es la función de activación aplicada. Sin perdida de generalización, esta formulación puede ser aplicable a otras dos capas cualesquiera.

En la Figura 2.26, la capa a la que se le asignan los datos (X) de entrenamiento se conoce como capa de entrada o *input*, la capa que predice los resultados finales es la capa de salida o *output* (Y) y las posibles capas que hay entre medias se conocen como capas ocultas o *hidden*.

Para calcular el error producido por cada variable de la red (pesos o *biases*), se deriva el error con respecto a cada elemento utilizando la regla de la cadena (ver ecuación 2.20). Esto se conoce como propagación hacia atrás o *backpropagation*.

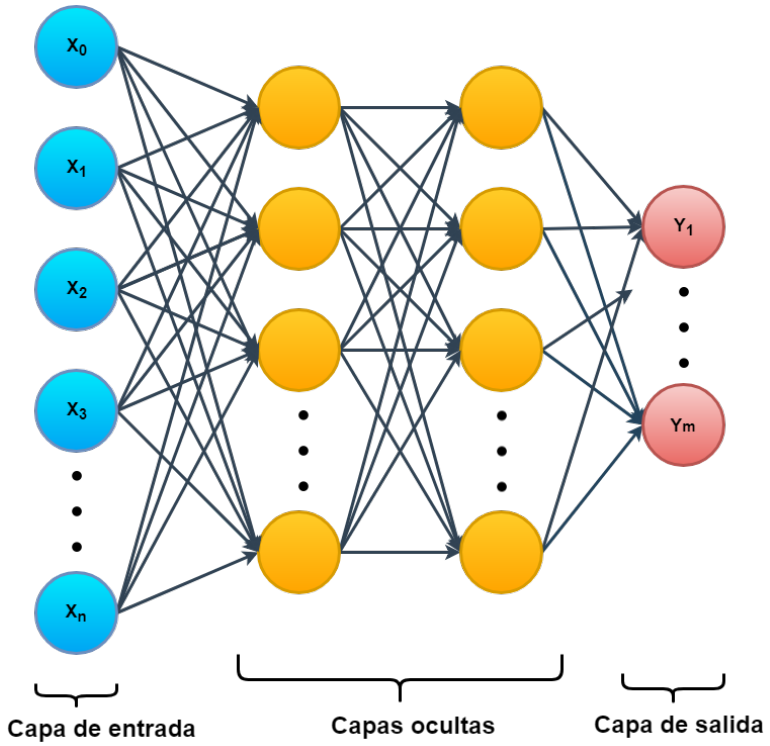


Figura 2.26: Ejemplo de una red *feed-forward* con dos capas ocultas

Así por ejemplo, para calcular el error producido por el vector de pesos $w^{(1)}$ con respecto a la salida $x^{(2)}$, primero se definiría el error, por ejemplo como $E = x^{(2)} - y$ y después se aplicaría la regla de la cadena sobre el parámetro en cuestión:

$$\frac{\partial E}{\partial w^{(1)}} = \frac{\partial E}{\partial x^{(2)}} \frac{\partial x^{(2)}}{\partial w^{(1)}}$$

Esto tiene el problema de que en redes muy profundas, las primeras capas pueden ser difíciles de entrenar, debido a que el gradiente que se propaga hacia atrás normalmente va siendo cada vez menor (aunque esto puede solucionarse en parte usando unidades ReLU [42]).

Estas redes *feed-forward*, sin embargo, no son apropiadas para tratar con señales temporales como pueda ser el sonido porque su estructura no es adecuada para señales de longitud variable. Para eso típicamente se utilizan las redes neuronales recurrentes.

2.8.1. Redes neuronales recurrentes

Las redes neuronales recurrentes (RNN) son redes realimentadas a lo largo del tiempo. Esta realimentación (o recurrencia) permite a la información persistir a lo largo del tiempo, de tal modo, que a la hora de calcular una salida, la red tendrá en cuenta eventos que ya haya visto con anterioridad. Es decir, contará con un contexto.

Este tipo de redes son adecuadas para trabajar con secuencias de datos, como pueden ser datos procedentes de sonido, la bolsa, escritura, los genes, etc.

La Figura 2.27 muestra una representación gráfica de estas redes. La red se alimenta con sucesivas entradas en el tiempo $(X_0, X_1, ...)$ y devuelve para cada una una salida $(Y_0, Y_1, ...)$. La recurrencia mencionada se obtiene al conectar la celda mediante la matriz W con ella misma, siendo el estado de la celda el que permite mantener el contexto a lo largo del tiempo.

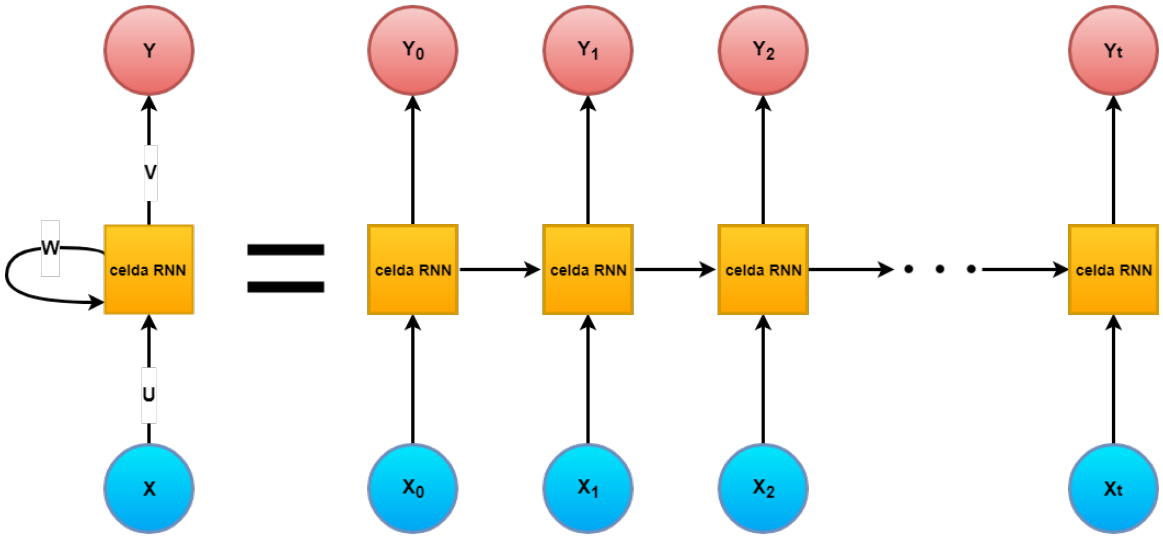


Figura 2.27: Representación de una RNN. A la izquierda, la red enrollada con las matrices que componen esta estructura (U, WyV). A la derecha, la RNN desenrollada en el tiempo.

El estado en cada instante temporal s_t se define como:

$$s_t = f_s(Ux_t + Ws_{t-1})$$

y la salida y_t en cada instante se calcula como:

$$y_t = f_y(Vs_t),$$

dónde f_y y f_s son las funciones de activación elegidas para la salida y el estado, respectivamente, y U, V y W son las matrices que conectan la entrada x con el estado temporal s_t y con la salida y_t tal y como se muestran en la figura 2.27.

Este tipo de redes, sin embargo, presenta algunos problemas que pueden dificultar bastante el aprendizaje. Es computacionalmente muy caro mantener el contexto durante mucho tiempo porque para calcular el gradiente hay que aplicar *backpropagation* a lo largo del tiempo, y debido a esto, es probable que los gradientes calculados sean muy pequeños para los primeros instantes de tiempo debido a la regla de la cadena (ecuación 2.20). Esto implica que las RNN tendrán problemas para aprender dependencias que estén muy separadas en el tiempo.

$$\frac{\partial g(f(x))}{\partial x} = \frac{\partial g(f(x))}{\partial f(x)} \frac{\partial f(x)}{\partial x} \tag{2.20}$$

Propagación hacia atrás a través del tiempo

La propagación hacia atrás en el tiempo es el algoritmo utilizado para entrenar las redes neuronales recurrentes. Es similar a la propagación hacia atrás de las redes *feed-forward*, pero aplicado a una RNN desenrollada como se muestra en la Figura 2.27.

Definiendo el error como $E_t = -ylog(\hat{y})$ (entropía cruzada), donde $\hat{y}$ es la salida de la red (también llamada estimación) e y el valor correcto de $\hat{y}$, y aplicando este procedimiento para las matrices W, U y V definidas en la Figura 2.27, aplicando la regla de la cadena (ecuación 2.20) sus gradientes se definen como:

$$\frac{\partial E_t}{\partial U} = \frac{\partial E_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial z_t} \frac{\partial z_t}{\partial s_t} \frac{\partial s_t}{\partial U} \tag{2.21}$$

$$\frac{\partial E_t}{\partial V} = \frac{\partial E_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial z_t}\frac{\partial z_t}{\partial V} \tag{2.22}$$

$$\frac{\partial E_t}{\partial W} = \frac{\partial E_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial z_t}\frac{\partial z_t}{\partial s_t}\frac{\partial s_t}{\partial W} \tag{2.23}$$

donde z_t es la salida de la red sin la función de activación (es decir $z_t = s_t V$) y s_t el estado de la red.

Problema del desvanecimiento del gradiente (*vanishing gradient*)

Este problema fue originalmente descrito por Sepp Hochreiter [2]. Los gradientes para U y W, ambos aplican la regla de la cadena sobre el estados actual s_t. Como s_t en realidad depende también del estado anterior s_{t-1} y este a su vez del anterior y así hasta llegar al estado s_0, esta regla de la cadena es un multiplicatorio sobre todos los instantes temporales que puede expresarse como:

$$\frac{\partial s_t}{\partial W} = \frac{\partial s_t}{\partial s_{t-1}}\frac{\partial s_{t-1}}{\partial s_{t-2}}\cdots\frac{\partial s_0}{\partial W} = (\prod_{t_m=0}^{t-1}\frac{\partial s_{t_m+1}}{\partial s_{t_m}})\frac{\partial s_0}{\partial W} \tag{2.24}$$

Para el caso de U esto es incluso más complicado porque el estado anterior s_{t-1} depende no sólo de s_t, sino también de U. Detalles sobre su cálculo puede encontrarse en [43].

Cada uno de los términos $\frac{\partial s_{t_m+1}}{\partial s_{t_m}}$ presentes en la ecuación 2.24 es, por lo general mucho menor que 1, por lo que el multiplicatorio $\prod_{t_m=0}^{t-1}\frac{\partial s_{t_m+1}}{\partial s_{t_m}}$ para t altos tenderá a ser cero. Esto se conoce como problema del desvanecimiento del gradiente o *vanishing gradient problem* y es la causa de que las RNN no puedan aprender dependencias muy largas en el tiempo. Las redes *feed-forward* también son susceptibles de encontrarse con este problema si son muy profundas.

Para solucionar estos problemas existen variantes de estas redes, las cuales contienen puertas o *gates* para controlar qué información pasa, sale o se borra del estado de la red. Existen principalmente dos estructuras LSTM y GRU.

2.8.2. Redes de gran memoria de corto plazo

Las redes de gran memoria de corto plazo o redes *Long Short-Term Memory* (LSTM) abordan el problema del desvanecimiento del gradiente mediante la inclusión de una celda de memoria explícita C_t y distintas puertas para acceder a ésta. Estas redes aparecieron por primera vez en 1997 [2] como solución a los problemas mencionados anteriormente, y en los años siguientes se fueron desarrollando sucesivas mejoras, como la adición de *forget gate* [44] (puerta que permite olvidar la información de la celda) o *peepholes connections* (conexiones que permiten a las capas de las puertas tener en cuenta la celda de memoria).

Esta estructura contiene por lo general tres puertas o *gates* para introducir, olvidar u obtener información de la celda de memoria (ver Figura 2.28).

La puerta de entrada (*input gate*) se encarga de decidir qué información debe entrar en la celda C_t. Para esto se calcula el vector i_t, de las mismas dimensiones que x concatenado con la salida anterior h_{t-1} y con valores comprendidos entre 0 y 1 indicando para cada valor de este vector como de relevante es para entrar en C_t. Este vector se calcula como se muestra en la ecuación 2.25.

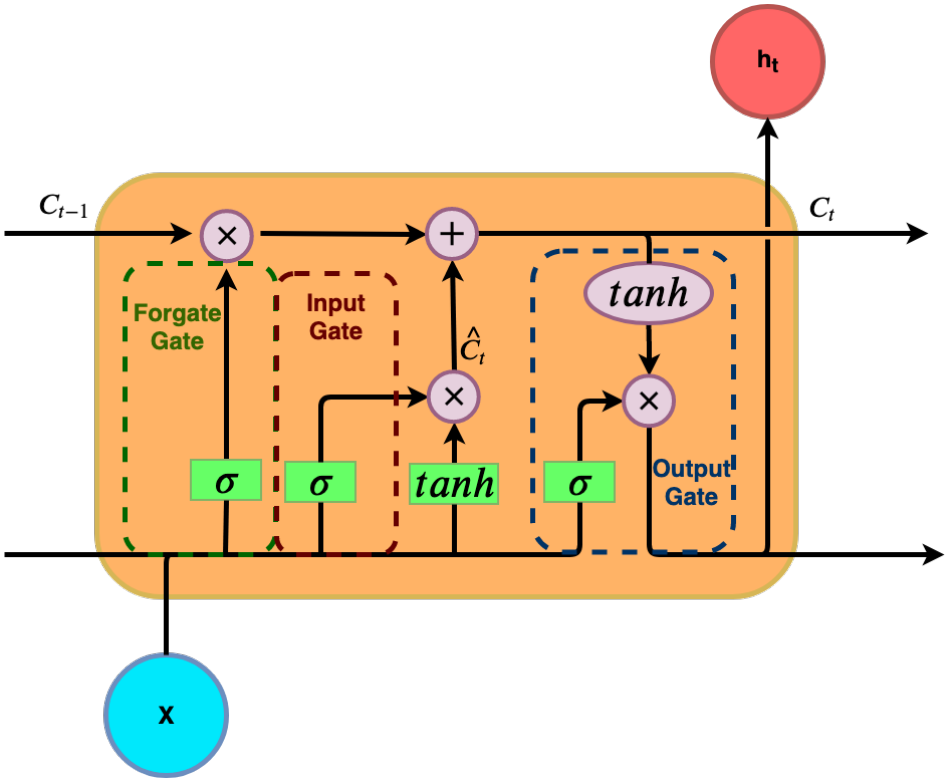


Figura 2.28: Representación de una celda *Long Short-Term Memory* con las distintas puertas que la componen.

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i). \tag{2.25}$$

Los candidatos a entrar en esta celda $\hat{C}_t$ se calculan como:

$$\hat{C} = tanh(W_c[h_{t-1}, x_t] + b_c),$$

donde *tanh* es la función tangente hiperbólica.

La LSTM también debe aprender a desechar información de la celda de memoria cuando considere que la información que hay en ella ya no es relevante. Para ello la puerta de olvido (*forget gate*) hace uso de la ecuación:

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f).$$

Esta ecuación dará también como salida un vector de la longitud de la celda de memoria, con valores comprendidos entre 0 y 1 expresando como de relevante es esa información. En este caso en particular, es importante iniciar el *bias* b_f a 1, ya que si no, la LSTM empezaría desechando toda la información y el aprendizaje sería mucho más lento.

Finalmente, la salida h_t de la celda vendrá determinada por la puerta *output gate* siguiendo la ecuación

$$h_t = o_t * tanh(C_t),$$

donde

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o).$$

2.8.3. Unidades recurrentes con puertas

Las unidades recurrentes con puertas o *Gated Recurrent Unit* (GRU) tratan de solventar el problema del desvanecimiento del gradiente de un modo distinto y más simple que las LSTM. No poseen una celda de memoria separada como era el caso de C_t para las LSTM (esta información viaja directamente en h_t, que es a su vez la salida de la red), pero al igual que éstas, también tienen distintas puertas para controlar el flujo de información. Se muestra su estructura en la Figura 2.29.

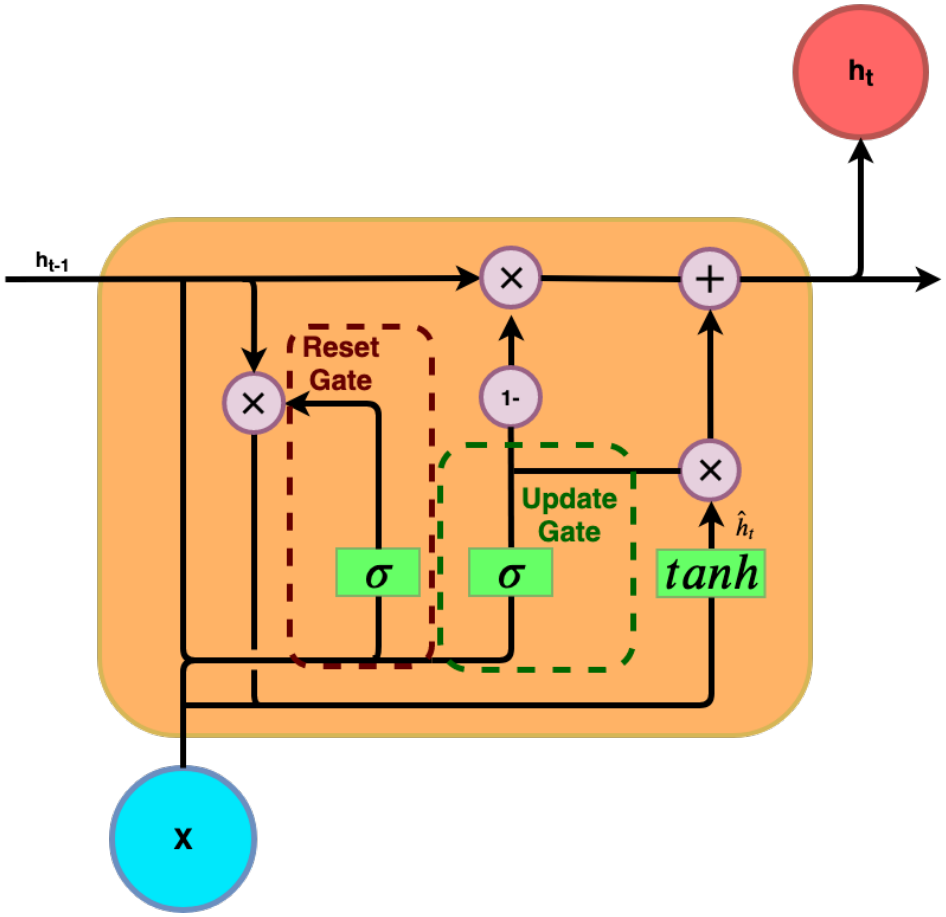


Figura 2.29: Representación de una celda *Gated Recurrent Unit* con las distintas puertas que la componen.

Estas estructuras fueron inventadas en 2014 [45], bastante más tarde que las LSTM. Cuentan con la ventaja de ser computacionalmente más simples al contar con menos puertas y aún así su rendimiento puede equipararse al de las LSTM en muchos casos [46][47].

En las unidades GRU, la puerta *update* puede ser vista como la puerta *forget* junto con la *input* de las LSTM. Esta usa x_t y h_{t-1} para calcular un vector de valores comprendidos entre 0 y 1 que reflejará qué datos de h_{t-1} deben ser olvidados. La expresión de la puerta *update* o de actualización viene dada por:

$$u_t = \sigma(W_u[h_{t-1}, x_t] + b_u). \tag{2.26}$$

Este mismo vector se usará después para seleccionar los valores de h_{t-1} candidatos a entrar en la memoria de la celda.

La puerta *reset* se encargará de determinar, usando x_t y h_{t-1}, qué información de h_{t-1} no ha de ser tenida en cuenta para calcular los nuevos valores candidatos a la memoria $\hat{h}_t$. Ésta se

calcula mediante la ecuación

$$r_t = \sigma(W_r[h_{t-1}, x_t] + b_r). \tag{2.27}$$

Finalmente, los valores candidatos a entrar en memoria se calcularán como:

$$\hat{h_t} = tanh(W[r_t * h_{t-1}, x_t] + b)$$

donde *tanh* es la función tangente hiperbólica, r_t es el vector definido en la puerta reset y W y b los pesos y *bias* de esta puerta.

Tanto las celdas LSTM como las GRU son dos soluciones válidas para combatir el problema del desvanecimiento del gradiente. Dependiendo del tipo de problema y de la potencia de computación disponible serán más adecuadas unas u otras. En este trabajo exploraremos cómo afectan ambas al problema del reconocimiento de fonemas.

3

Metodología y Desarrollo

3.1. Introducción

Este capítulo profundizará sobre el propio desarrollo del trabajo. Para ello se mencionarán las librerías usadas y de qué equipos disponemos.

Seguidamente se explicará la estructura de la base de datos de TIMIT, de qué modo están relacionados sus archivos y qué clases estaremos interesados en reconocer. Después, se realizará un análisis exploratorio de los datos para comprender mejor su distribución y características relevantes. Tras esto, se extraerán características usando para ello extractores predefinidos y extractores de características con aprendizaje no supervisado y finalmente se propondrá la estructura de un clasificador de características el cual se entrenará con las características de los extractores antes mencionados.

La Figura 3.1 muestra un diagrama de flujo indicando los pasos principales que se van a realizar en el trabajo. En la figura hay tres partes diferenciadas, la primera de ellas se corresponde con la extracción de las características de la señal de audio. El siguiente paso, que lo usaremos para visualizar las características, es lo correspondiente al círculo identificado con PCA. Hacemos uso de PCA como método de reducción y para realizar diagramas de representación con las componentes principales. Por último, la línea que desemboca en la predicción, clasifica los fonemas haciendo uso de una red profunda LSTM.

3.2. Librerías y Equipos

La implementación del trabajo desarrollado, en su totalidad, ha sido realizado en *python*. Se ha utilizado el framework de *tensorflow* [48] para el diseño de modelos de redes neuronales, como pueden ser las redes recurrentes y las máquinas de Boltzmann, usando además *numpy* y *pandas* para el tratamiento de los datos y parte de algoritmia. Los *scripts* creados para este trabajo pueden ser obtenidos en `https://github.com/isaacgg/TFM`.

En el momento de comenzar este trabajo, quizás los principales frameworks para trabajar con *machine learning* sobre GPU eran *tensorflow*, *keras* [49], *theano* [50] y *pytorch* [51] en versión beta. Se ha elegido *tensorflow* porque permite un desarrollo a más bajo nivel que *Keras*, algo

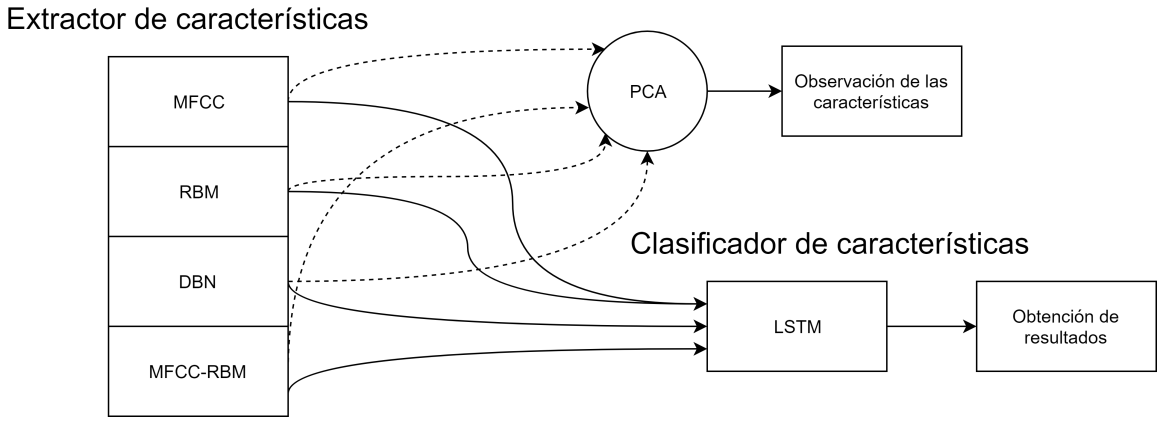


Figura 3.1: Esquema de los distintos pasos que se llevarán a cabo en el reconocimiento de fonemas en este trabajo.

que era fundamental para el estudio realizado con las RBM, ya que *Keras* no permite de un modo fácil implementar arquitecturas que no estén ya definidas en su librería.

En cuanto a las máquinas utilizadas, la parte más ligera (extracción de características clásicas y entrenamiento de la RBM) se realiza en un ordenador con una tarjeta *nVidia* GTX960M de 2GB y 16 GB de RAM. Para el entrenamiento de las RNN elegimos la plataforma de *Floydhub*[52] porque su curva de aprendizaje es muy simple. En esta plataforma contamos con una tarjeta Tesla K80 de 12GB y 61 GB de RAM a un precio de 1.2$/hora.

3.3. Preparación de los datos en TIMIT

La base de datos de TIMIT se presenta con una estructura jerárquica de carpetas, separando en el primer nivel los conjuntos de *test* y *train* sugeridos, las distintas regiones geográficas en el siguiente nivel y los distintos hablantes en el último. Finalmente los distintos archivos de audio en los que habló cada persona van precedidos de "SA", "SI" o "SX" según del tipo de frase que se trate (como se explica en la sección 2.3) y acabados en un número para identificar únicamente a cada frase. Junto con éstos existen otros archivos con el mismo nombre pero con extensión .PHN, .WRD o .TXT. La Figura 3.2 muestra un ejemplo de la jerarquía de directorios y ficheros en la base de datos. En este caso se parte del directorio TRAIN, región geográfica identificada como DR1 e individuo FCJF0. En este último directorio se encuentran los archivos de audio, por cada frase hay cuatro archivos con extensiones .wav, .PHN y .TXT.

Para este trabajo, sólo será de nuestro interés los archivos .wav y los archivos .PHN que contiene la segmentación en fonemas para su archivo .wav homónimo. En estos archivos .PHN la segmentación viene marcada con el *frame* de inicio, el *frame* de fin y el fonema presente entre estos dos, como se muestra en el ejemplo siguiente:

```
    0 2260 h#
2260 2730 d
2730 4120 uh
4120 4600 n
4600 6864 ae
...
```

Como se menciona en la sección 2.3, es típico reducir las clases de los fonemas de 61 a 39 (ver tabla 2.2) agrupando así las clases que son similares entre ellas. En este trabajo aplicaremos esta misma reducción, quedándonos después con las siguientes clases a clasificar: /aa/, /ae/, /ah/,

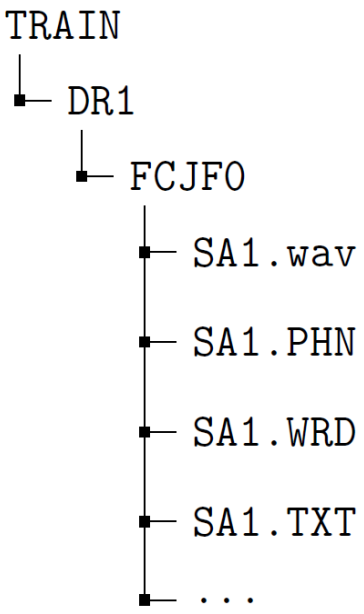


Figura 3.2: Estructura de carpetas jerárquica encontrada en la base de datos de TIMIT.

/aw/, /ay/, /b/, /ch/, /d/, /dh/, /dx/, /eh/, /er/, /ey/, /f/, /g/, /h/, /ih/, /iy/, /jh/, /k/, /l/, /m/, /n/, /ng/, /ow/, /oy/, /p/, /r/, /s/, /sh/, /sil/, /t/, /th/, /uh/, /uw/, /v/, /w/, /y/, /z/.

Además, dado que las frases marcadas con "SA" son las mismas para todos los hablantes, (tanto del conjunto de *train* como el de *test*) se descartan del conjunto de *test* para evitar que los resultados sean demasiado optimistas.

3.4. Análisis exploratorio de los datos

Antes de enfrentarse a cualquier problema de análisis de datos, suele ser conveniente realizar un análisis exploratorio de los mismos (*Exploratory Data Analysis* en inglés o EDA por sus siglas). La finalidad de este análisis es comprender mejor los datos y analizar qué características de éstos pueden ser más relevantes, de esta forma, podemos estudiar mejor cuáles son las características más discriminantes.

Para ayudarnos en este paso, haremos uso de la biblioteca *pandas* de *python*. Esta librería imita en parte el comportamiento del lenguaje *R* y es ampliamente usada en el campo del análisis de datos. Con ella, se pueden organizar los datos en *DataFrames*, que son estructuras que constan de varias columnas para guardar las diferentes propiedades de los datos y diferentes filas para guardar cada uno de los datos con sus características.

Como 39 categorías son muchas para representar claramente en un gráfico, se puede hacer una categorización según algunas de las características explicadas en la sección 2.2. Existen algunas clasificaciones ya creadas[53][1]. Para este trabajo usaremos la mencionada en [1], reduciendo así los fonemas a vocales, fricativos, *stops*, nasales o silencios, tal y como se representa en la Tabla 3.1.

En primer lugar, conviene echar un vistazo a la distribución de las clases de los fonemas. Como se puede comprobar en la Figura 3.3, se dispone de un dataset con un conjunto de clases desbalanceado, con la clase *sil* como clase mayoritaria. Como se puede ver en la tabla 2.2, esta

Tabla 3.1: Clasificación de los distintos fonemas por categoría, tal y como se describe en [1].

vocales	/aw/, /ay/, /uh/, /r/, /w/, /oy/, /er/, /ae/, /ih/, /iy/, /l/, /ow/, /ah/, /eh/, /y/, /aa/, /uw/, /ey/
fricativos	/h/, /sh/, /z/, /f/, /v/, /dh/, /s/, /th/
stops	/t/, /g/, /k/, /jh/, /b/, /d/, /p/, /ch/
nasales	/m/, /n/, /ng/
silencios	/dx/, /sil/

clase incluye no sólo los silencios, sino también el cierre de algunos fonemas oclusivos. Se aprecia también que la distribución de clases es muy similar para ambos conjuntos de *test* y *train*.

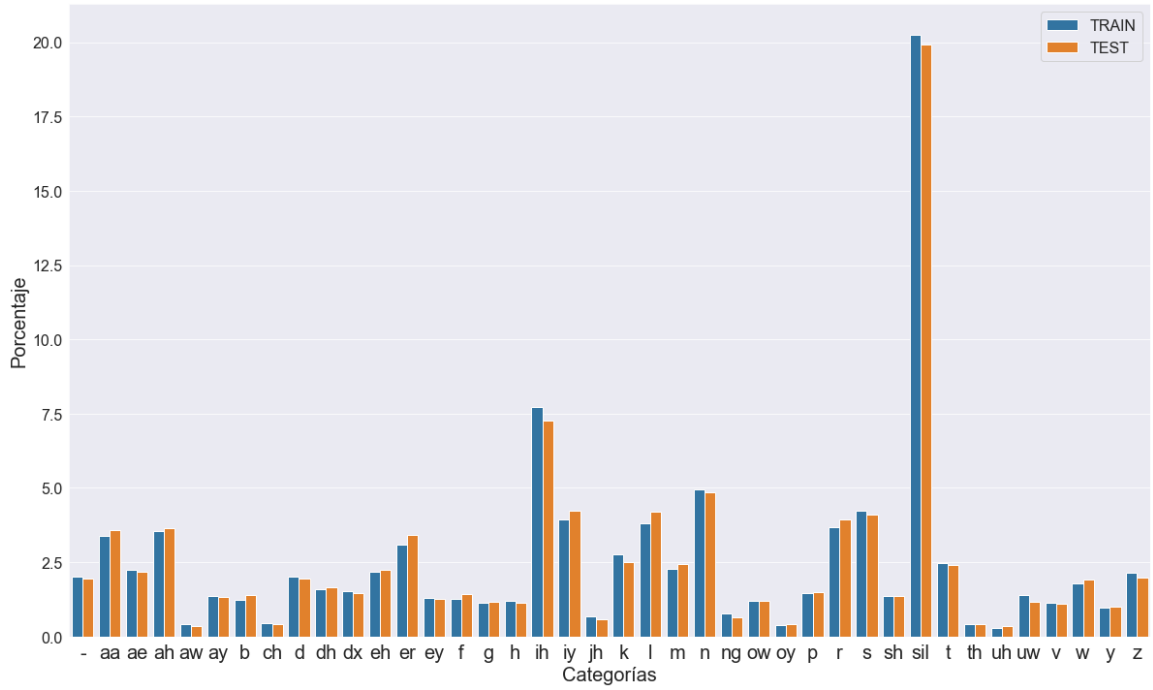


Figura 3.3: Distribución de fonemas en TIMIT para los conjuntos de *train* y *test*. El histograma está normalizado en ambos conjuntos: entrenamiento y test.

En cuanto a las características propias de las señales, la primera que puede ser interesante de analizar es la duración de cada fonema. La Figura 3.4 muestra un gráfico de violín con esta información. Los gráficos de violín son similares a un *boxplot*, pero mostrando además estimaciones de densidad kernel. Consideramos este tipo de gráfico el adecuado porque nos permite observar si cada categoría es o no multimodal. Dado que las categorías definidas constan de varias clases de fonemas (ver Tabla 3.1), es probable que esto ocurra. Se han descartado los fonemas mayores de 8000 *frames* para una mejor visualización, porque estos fonemas son los silencios que se producen antes de que el hablante empiece a hablar o desde que acaba de hablar hasta que se apaga el micrófono. Téngase en cuenta que estos fonemas en propiedades como la longitud provoca que el gráfico se vea pequeño y achatado. Se puede apreciar en la Figura 3.4 que los fonemas varían en cuanto a longitud, pero los más cortos tienden a ser *stops*. Esto tiene sentido cuando se comprueba qué fonemas corresponden a esta categoría (/t/, /p/, /b/...). Además, también se aprecia que los silencios tienen una distribución multimodal, con dos subdistribuciones bien diferenciadas. La Figura 3.5 desgrana esta categoría en sus distintas clases del conjunto original de 61 fonemas. Se observa que los fonemas /h#/ y /pau/ tienen unos valores bastante alejados del resto de su clase, siendo éstos los que crean esa segunda

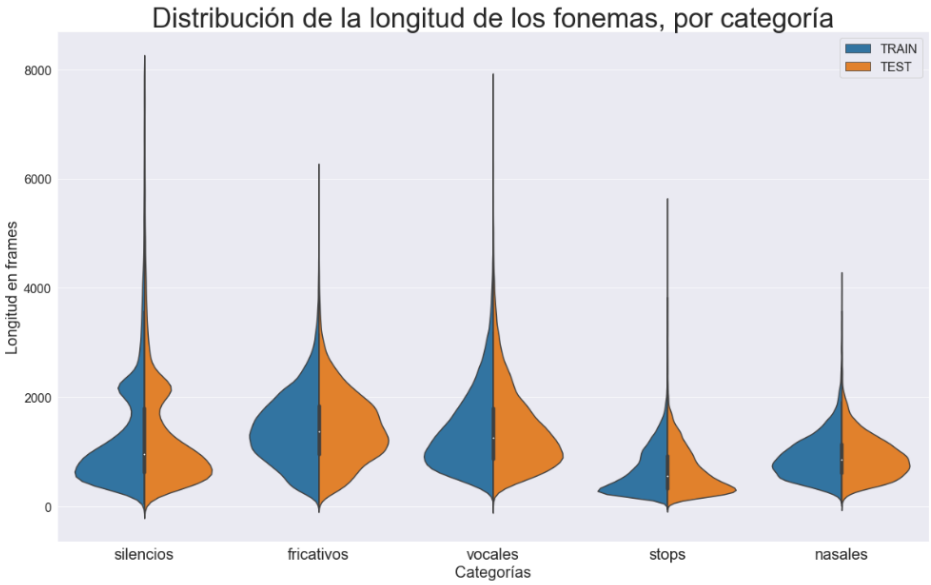


Figura 3.4: Duración de fonemas en TIMIT, agrupados éstos por la categoría a la que pertenecen. Los datos están acotados a un máximo de 8000 *frames* para una mejor visualización. La representación se ha realizado usando gráficos de violín [54].

subdistribución que se observaba en la Figura 3.4. Estos fonemas corresponden a los silencios al comienzo y al acabar la frase (/h#/) y a las pausas dentro de la misma (/pau/).

Otra característica importante que caracteriza la señal es su energía. De hecho, en ocasiones, es común reemplazar el primer coeficiente de los parámetros extraídos con MFCC por el logaritmo de la energía. Ésta se calcula como $\frac{\sum(|x|^2)}{N}$, donde x es la señal dada y N es la longitud de la misma. Intuitivamente, es de esperar que los silencios sean la categoría con menor energía, mientras que los vocales, por contra, tendrán una energía más alta. Esta distribución se muestra en la Figura 3.6, donde se representa el logaritmo de la energía por cada fonema.

Por último, una característica que debería ser discriminante es la frecuencia máxima de la señal. Para encontrar esta frecuencia, se aplica la FFT a cada ventana de señal y se busca el *bin* mayor. Se le aplica la ventana de Hann para evitar la fuga espectral explicada en la sección 2.4.2. Además, sabiendo que se utiliza una FFT de 512 puntos y que la señal está muestreada a 16 KHz, cada *bin* se corresponde a $\frac{16\,KHz}{512} = 31{,}15\frac{Hz}{bin}$. Esta distribución se muestra en la gráfica 3.7. Se observa que los fonemas que tienen sus frecuencias más definidas son los vocálicos y los nasales, por contra, los fricativos son los más ruidosos al no tener una frecuencia clara definida.

Al igual que ocurría con la longitud para los silencios, se intuyen varias distribuciones muy diferenciadas para los fonemas fricativos. Desgranando esta categoría por fonemas (ver Figura 3.8), podemos comprobar que efectivamente es así, ya que los fonemas como /s/, /z/ o /sh/ tienden a ser los más ruidosos. Además, es posible apreciar en la Figura 3.8 que hay varios fonemas con distribuciones bimodales.

3.5. Extracción de características de la señal

3.5.1. Enventanado

Como se mencionó en la sección 2.4, las secciones de ventanas suelen de ser de entre 20 y 40 milisegundos aproximadamente. Para este trabajo, se ha elegido un tamaño de ventana

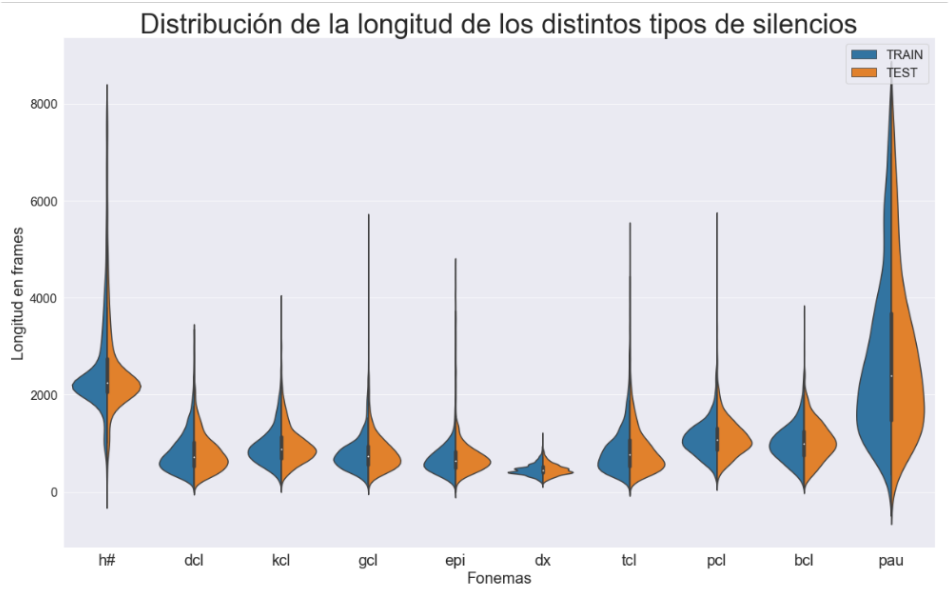


Figura 3.5: Gráfico de violín para analizar la duración de los distintos fonemas de silencio. Los datos están acotados a un máximo de 8000 *frames* por fonema para una mejor visualización.

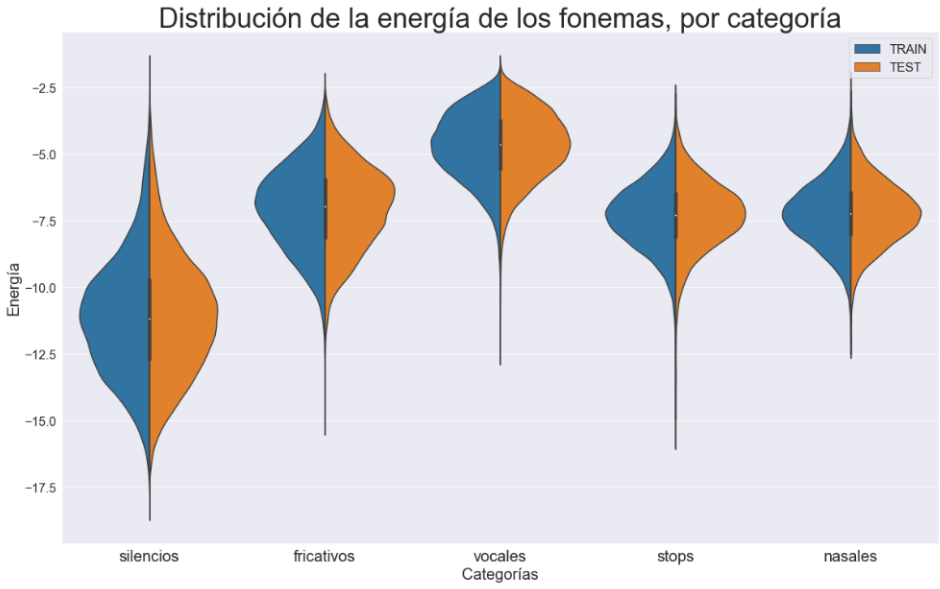


Figura 3.6: Gráfico de violín para analizar las distintas energías de los fonemas. Se está representando la distribución del logaritmo de la energía por cada fonema para una mejor visualización.

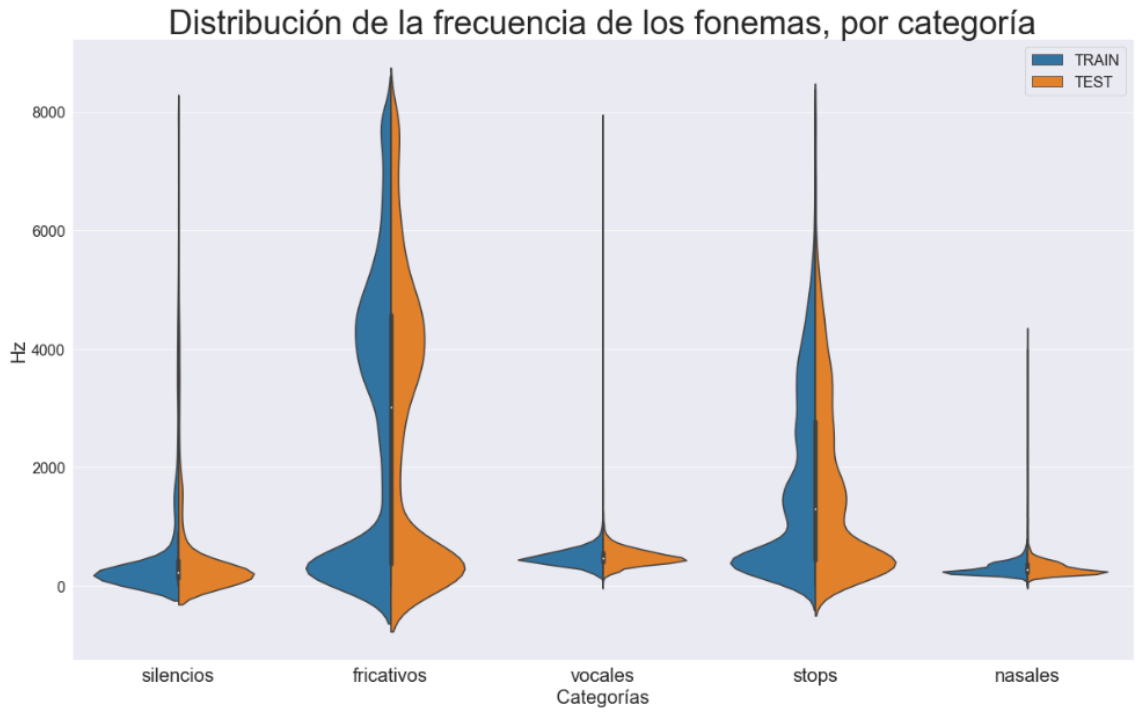


Figura 3.7: Gráfico de violín con las frecuencias dominantes de los fonemas. Se aprecia como los fonemas fricativos son los más ruidosos (sus frecuencias máximas están dispersas a lo largo del espectro).

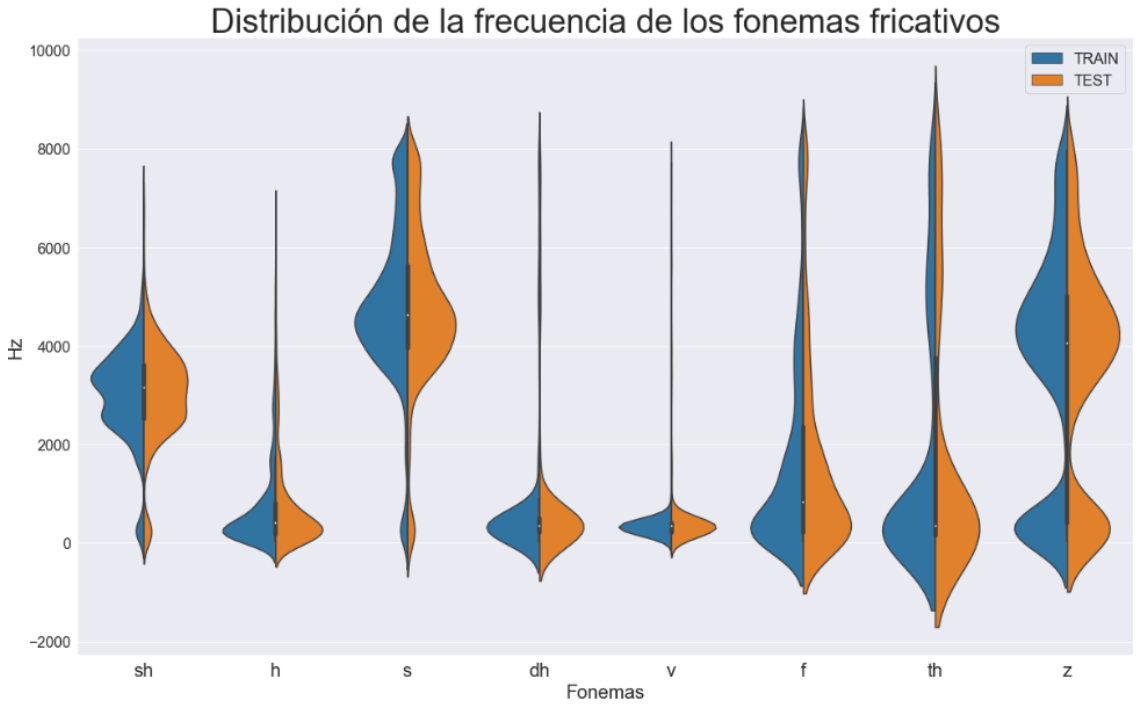


Figura 3.8: Gráfico de violín con las frecuencias de los fonemas fricativos. Puede verse como los fonemas /s/ y /z/ son los más ruidosos. Fonemas como /v/ o /dh/ tienen sus frecuencias más localizadas.

de 25 milisegundos y un *offset* de 10 milisegundos. Sabiendo que los audios de TIMIT están muestreados a una frecuencia de 16 KHz, esto se traducirá en que las ventanas serán de 400 *frames* y el offset entre ventanas de 160 *frames*. Además se le aplicará la ventana de Hann (mostrada en la Figura 2.8) a cada ventana.

3.5.2. Extractores de características seleccionados

En esta sección se proponen los extractores de características a usar en este trabajo. Más adelante, se estudiarán estas características y se utilizarán estos extractores en la clasificación de fonemas. Se van a explorar cuatro enfoques:

- MFCC como método predefindo clásico de extracción de características.
- *Restricted Boltzman Machine* como método que aprende a partir de los datos.
- *Deep Belief Networks* que consiste en una versión apilable de las RBM.
- Una combinación de características clásicas MFCC con RBM para aprender a extraer características de estos datos.

Extractor de características MFCC

Para la extracción de características con MFCC, se propone una estructura como sigue (ver Figura 2.12):

1. Enventanado de 25 ms y 10 ms de *offset* con ventana de Hann
2. FFT de 512 puntos
3. 26 bancos de filtros definidos entre 0 y 8 KHz
4. Reducción a 13 componentes con DCT
5. Sustitución del componente 0 de la DCT por el logaritmo de la energía

Esta estructura es quizás la más típica cuando se emplean características MFCC en sistemas de *speech recognition*. Para extraer estas características se utilizó la librería *python speech features*.

Extractor de características con RBM

Para nuestro extractor de características con RBM utilizaremos también ventanas de 25 ms y 10 ms de *offset* con ventana de Hann. Para poder adaptar la señal a las ondas de sonido a la entrada, usaremos activaciones gaussianas para las unidades visibles. La señal de audio ha de ir normalizada a media 0 y varianza unidad. Este tipo de unidades son las más adecuadas porque podemos suponer que por el comportamiento sinusoidal de las ondas que forman el sonido, sus amplitudes siguen una distribución gaussiana de media 0.

En cuanto a las unidades ocultas de la RBM usamos unidades ReLU porque han demostrado su eficacia para la extracción de características en señales de audio [32][33]. El código implementado para esta red puede encontrarse en `https://github.com/isaacgg/TFM/blob/master/commons/rbms/GaussReLUrbm.py`.

La RBM diseñada contará con 400 unidades visibles (correspondientes a los 25ms de entrada de audio) y 400 unidades ocultas. Cada una de estas unidades ocultas se corresponderá con una característica que buscará la RBM en cada ventana de audio. Estas características serán las activaciones de las unidades ocultas cuando las unidades visibles se alimentan con una ventana de audio.

Finalmente esta red se entrena aplicando un descenso por gradiente estocástico con un *learning rate* (LR) de 0.1 y 0.5 de *momentum* en *batches* de 128 ventanas de señal.

Extractor de características con *Deep Belief Networks*

Una *Deep Belief Network* (DBN) consiste en apilar varias RBM entrenadas normalmente de un modo *layerwise*, es decir, capa a capa. Esto quiere decir que las características calculadas por la primera RBM se utilizarán para entrenar la segunda capa de la RBM. Esta estructura es supuestamente capaz de encontrar características más profundas de la señal que podrían dar lugar a un mejor resultado. El código para esta red es el mismo que para las RBM para la primera capa. Para la segunda capa se empleará el código `https://github.com/isaacgg/TFM/blob/master/commons/rbms/ReLUReLUrbm.py` utilizando como entrada las características RBM calculadas anteriormente.

Para entrenar esta segunda red, por tanto, debemos usar unidades ReLU como activaciones de las unidades visibles, ya que estas unidades deben ser capaces de adaptarse a la distribución de salida de la primera capa. Para las unidades ocultas, las unidades típicas que podríamos usar son Bernoulli (también llamadas binarias) o ReLU de nuevo. En este caso utilizaremos unidades ReLU.

Los parámetros escogidos para entrenar la segunda capa de la DBN son *learning rate* de 0.01, 0.5 de *momentum* y 90 épocas.

Combinando extractores de características

Por último, probaremos la combinación de extractores de características. Para ello, utilizaremos las características extraídas con MFCC en la entrada de una RBM. Esta RBM por tanto tendrá 13 unidades visibles (ya que como se explica en la sección 2.5.3, es el procedimiento habitual). Estas unidades, de nuevo, deberán ser adaptadas con activaciones gaussianas porque es el tipo de distribución que mejor se adaptará a esta entrada. En cuanto a las unidades ocultas elegiremos activaciones ReLU porque parecen dar un mejor resultado. Los parámetros para el entrenamiento de esta capa son iguales que los utilizados con RBM. Para calcular estas características se utilizó el mismo código que en las RBM, pero a partir de las características MFCC extraídas anteriormente.

3.6. Clasificadores de características

El siguiente paso es utilizar las características extraídas anteriormente con algún modelo para clasificar los fonemas contenidos en la señal. Estos modelos serán redes neuronales recurrentes con unidades LSTM, GRU o *vanilla* (es decir, RNN sin estructura compleja en la celda). Todos estos modelos compartirán la misma estructura de capas para poder hacer una comparación más justa. La estructura elegida es una RNN bidireccional de dos capas, cada capa consta de 128 neuronas en cada dirección (hacia delante y hacia atrás), conectada su salida con una matriz de pesos de 256 con una función *softmax* [55] a la salida (ver Figura 3.9).

Esta función *softmax* es ampliamente utilizada en problemas de clasificación, ya que transforma las salidas de las neuronas de las últimas capas de la red en una probabilidad entre 0 y 1, asegurando que la suma de todas ellas sea 1. Ésta se define como:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

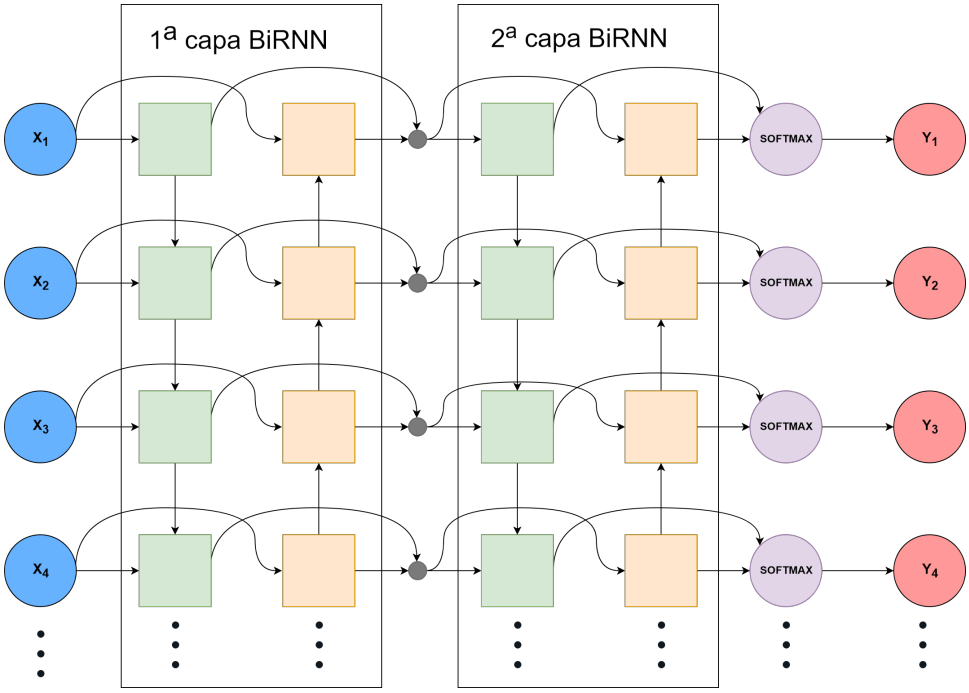


Figura 3.9: Representación de la arquitectura usada en este trabajo. Las cajas verdes representan las conexiones hacia delante, las naranjas las conexiones hacia atrás y el circulo gris la concatenación de las características de entrada.

Finalmente la salida de esta red es calculada con *Connectionist Temporal Classification* (CTC) [56][57]. CTC es un algoritmo que no trata de aprender dónde está la frontera en cada fonema, si no que los fonemas calculados sean los correctos. Usa para ello una etiqueta especial (que llamaremos "espacio en blanco") de tal modo que la matriz de salida de la red tendrá el número de etiquetas que se desean clasificar más una para el espacio en blanco (39+1, en nuestro caso). El espacio en blanco sirve para no clasificar un *frame* en cualquiera de las etiquetas de entrenamiento (ninguno de los 39 fonemas en nuestro caso). Por lo tanto, una vez finalizado el entrenamiento con CTC, la salida serán los fonemas clasificados, intercalados con espacios en blanco. Por último, para obtener los fonemas que componen las señal, el algoritmo CTC elimina los espacios en blancos y junta los fonemas adyacentes que son iguales, similar a como se muestra en la Figura 3.10.

Este algoritmo es el adecuado para este problema porque las fronteras entre cada fonema no son muy exactas, debido a que el etiquetado manual de la base de TIMIT tiene errores y además al enventanar la señal, esta frontera pierde todavía más precisión.

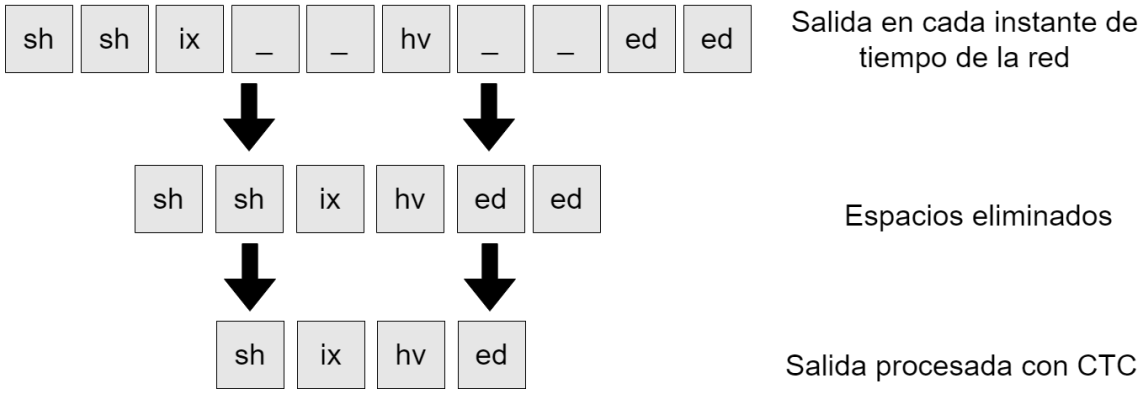


Figura 3.10: Representación del algoritmo CTC. El carácter "_" representa los espacios en blanco a la salida de la red. En el primer paso CTC elimina estos espacios. En el segundo junta las etiquetas contiguas iguales.

4
Resultados

4.1. Introducción

En esta sección se mostrarán los resultados que se obtienen con los métodos de extracción de características mencionados en el anterior capítulo. Con el propósito de visualizar las nuevas variables, será útil representar estas características en un espacio tridimensional. Para esto, haremos uso de *Principal Component Analysis*(PCA) [58]. PCA es una técnica ampliamente utilizada para la reducción de la dimensionalidad de los datos. PCA explica la mayor varianza posible de los datos con el menor número de dimensiones (o componentes principales) posible. El porcentaje de varianza total que explica PCA con las dimensiones a las que es reducido, da una idea de cuánta información contiene esta representación con respecto a la total. Con PCA, por tanto, reduciremos la dimensionalidad de estas características a 3, para poder ver su representación.

4.2. Características extraídas con MFCC

Para poder visualizar estas características, reducimos las 13 dimensiones iniciales de MFCC a 3 con PCA. Una vez reducidas a 3 podemos ver su representación en la Figura 4.1. En ésta, cada punto representa una ventana de señal, con el color correspondiente a la categoría a la que pertenece el fonema (o parte del fonema) presente en esa ventana. Con esta representación, estamos viendo un 46.5 % de varianza en la representación del panel izquierdo y un 41.1 % de varianza en el panel derecho. Debido a que el porcentaje de varianza explicado no es demasiado alto, no se puede confiar al 100 % en la representación mostrada, aunque sí puede ser válido para tener una idea de si el extractor tendrá resultados aceptables.

Analizando la figura, podemos ver una separación bastante buena entre fonemas fricativos, silencios y vocales (especialmente en la figura derecha). Sin embargo, en los fonemas nasales y sobre todo los *stops* no parece estar tan claro. Por las características analizadas en el EDA, sabemos que la energía es un factor relevante para discriminar los silencios y que la frecuencia hace lo propio con los fricativos (ya que éstos tienen componentes de muy alta frecuencia). Como MFCC separa los componentes en frecuencia (en los bancos de filtros) y además el primer componente de éstos ha sido sustituido por el logaritmo de la energía, no sorprende que sean

justo éstas (fonemas fricativos, silencios y vocales) las clases mejor separadas.

Figura 4.1: Representación de las características extraídas con MFCC después de reducir su dimensionalidad con PCA. Cada punto representa una ventana de señal distinta. Panel de la izquierda: muestra la representación de los fonemas en el espacio vectorial de las tres primeras componentes principales. Explica el 46.5 % de varianza. Panel de la derecha: muestra otra representación de los fonemas tomando las componentes principales 0, 3 y 4 explicando el 41.1 % de varianza.

4.3. Características extraídas con RBM

Después de completar el entrenamiento de las RBM gaussianas-ReLU es posible visualizar qué características ha aprendido a buscar esta red. Para ello, dibujamos las conexiones de cada unidad oculta con todas las visibles. Es decir, las características son en realidad los pesos que conectan a neuronas visibles y ocultas. Se puede decir, por tanto, que cada neurona oculta aprende a extraer una característica relevante de la señal. Algunos ejemplo de estas características se muestran en la Figura 4.2. Como se aprecia en esta figura, la RBM parece haber aprendido a buscar las componentes frecuenciales que componen a estas señales. Algunas de estas características parecen buscar componentes frecuenciales presentes en toda la ventana (primera característica de la Figura 4.2), mientras que otras parecen aprender a buscarlas localizadas sólo en una parte de la ventana (segunda característica de la Figura 4.2).

4.3.1. Reentrenando las unidades de *bias*

Aunque los pesos parecen haber aprendido a buscar buenas características, descubrimos que las unidades de *bias* ocultas tienden a ser muy negativas (ver Figura 4.3), provocando que en ocasiones la señal original no pueda ser recuperada[1] adecuadamente. A modo de ejemplo, en la Figura 4.4, el panel izquierdo superior muestra un fragmento de señal y como la RBM lo recupera adecuadamente. Sin embargo, el panel izquierdo inferior muestra un ejemplo de un fragmento de señal que la RBM no es capaz de recuperar. Como comprobaremos más adelante, este problema afectará negativamente al *accuracy* obtenido.

Para solucionar este problema, reentrenamos las unidades de *bias* (visibles y ocultas) con *backpropagation*. Este reentrenamiento no dejamos que afecte al resto de los pesos de la RBM, ya que las características aprendidas mediante el modelo basado en energía (ver ecuaciones 2.14,

[1]Trabajando con RBM, definimos la señal original como la ventana de audio que será la entrada a las unidades visibles de la RBM, y la señal recuperada como las unidades visibles pero muestreadas a partir de las unidades ocultas, usando la ecuación 2.12 (que a su vez han sido muestreadas a partir de la señal original usando la ecuación 2.11). Es decir, en la Figura 2.16 la señal original sería v para $t = 0$ y la señal recuperada sería v para $t = 1$.

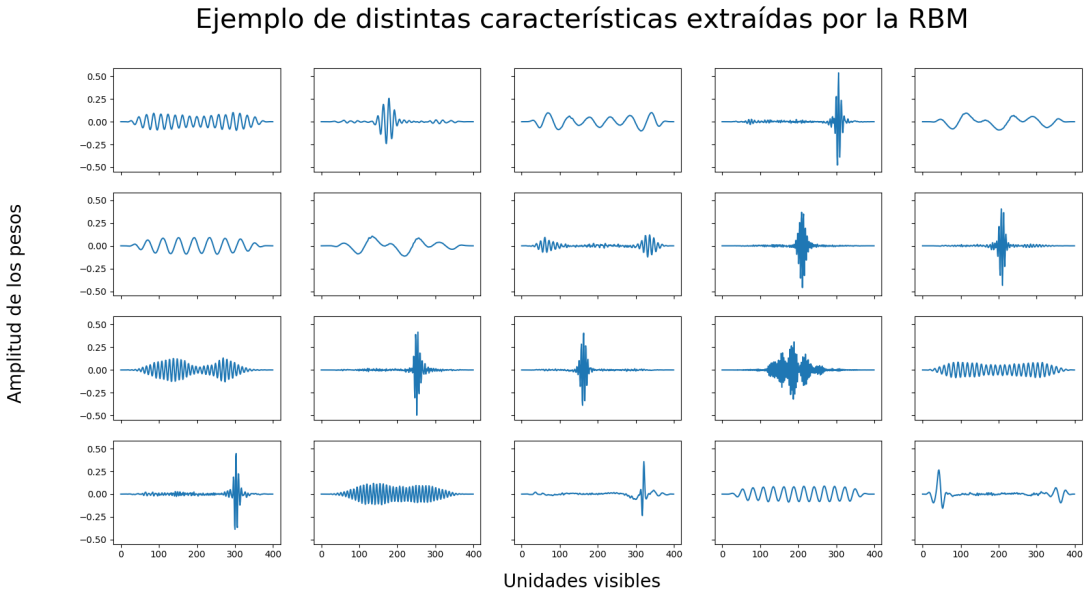


Figura 4.2: Ejemplo de algunas de las características aprendidas por la RBM gaussiana-ReLU. Estas características son los pesos que conectan a todas las unidades visibles con cada neurona oculta.

2.15 y 2.16) parecen ser buenas (por lo visto en la Figura 4.2). De tal modo que ahora, la definición de los gradientes de los *bias* será como:

$$\frac{\partial E}{\partial bias} = \frac{\partial RMSE(v, \hat{v})}{\partial bias}$$

para ambos bias (visibles y ocultos), donde $\hat{v}$ son las unidades visibles recuperadas, v son las unidades visibles originales y RMSE es el error cuadrático medio entre v y $\hat{v}$.

La Figura 4.5 muestra un histograma de la raíz del error cuadrático medio (RMSE por sus siglas en inglés) entre la señal original y la señal recuperada. Estas señales son todas las ventanas de los conjuntos de *test* y *train*. Se puede apreciar que tanto con el reentrenamiento de *bias* como sin él, muchas de estas ventanas son recuperadas adecuadamente (esto es, el *bin* correspondiente al 0 es el más alto), sin embargo, realizando el reentrenamiento de *bias* conseguimos reducir este error aún más, lo que más adelante ayudará a obtener mejores resultados. Se puede observar como el histograma está más desplazado hacia la izquierda (próximo a cero) cuando se realiza la corrección de reentrenar los *bias*.

Las características extraídas con este modelo pueden verse en la Figura 4.6. Se ha reducido la dimensionalidad de estas características con PCA para poder visualizarlas. En esta Figura, se aprecia que agrupa vocales y fricativos con bastante claridad, sin embargo, no se aprecia una separación clara entre nasales, silencios y *stops*. Por las características aprendidas (y mostradas en la Figura 4.2) observamos que esta estructura aprende a separar frecuencias. No es de extrañar, por tanto que las categorías que más distintas son en frecuencia (ver Figura 3.7) sean las que mejor separadas se muestran.

4.4. Características extraídas con DBN

Viendo los resultados arrojados por la RBM, se antoja necesario añadir algo más para tratar de separar algo mejor entre las otras clases. Por eso, a la entrada de la DBN, además de las

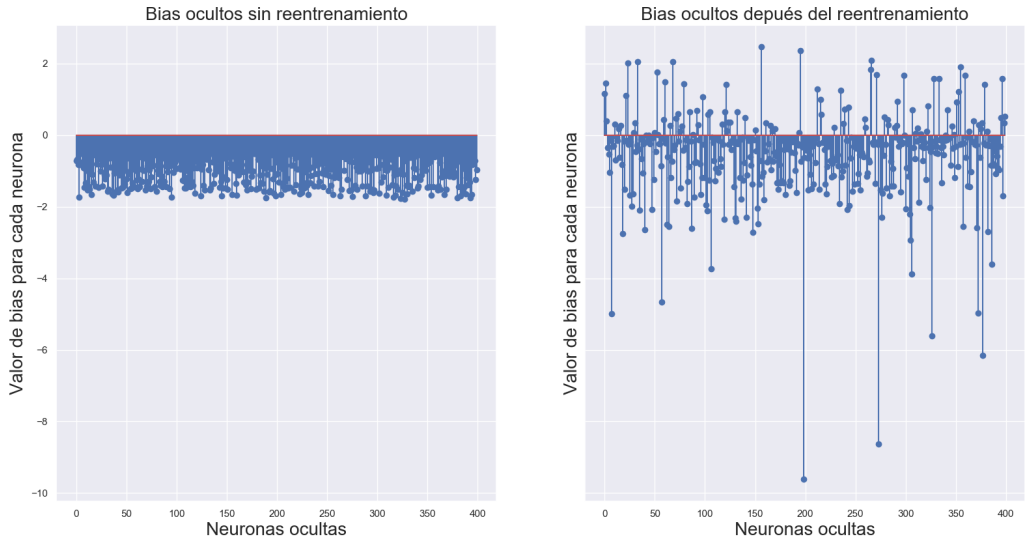


Figura 4.3: Unidades de bias. En el panel izquierdo unidades de *bias* sin reentrenamiento. En el panel derecho, unidades de *bias* después del reentrenamiento con *backpropagation*.

características extraídas por la RBM, le añadiremos el logaritmo de la energía de la ventana, del mismo modo que se hace con MFCC.

Al igual que para el resto de extractores, representamos las características extraídas por la DBN en la Figura 4.7. Éstas no parecen diferir mucho de las extraídas por la RBM. A pesar de haber añadido la energía, los silencios (que son la clase a la que más debería haber afectado este cambio por lo mostrado en la Figura 3.6), siguen sin verse agrupados.

4.5. Características extraídas con la combinación MFCC-RBM

Finalmente las características extraídas con MFCC-RBM se muestran en la Figura 4.8. Éstas son muy parecidas a las extraídas con MFCC, pero cuentan con un mayor número de dimensiones, (al ser la salida de la RBM mayor que la entrada) lo que en un principio podría ayudar a una mejor clasificación, siempre y cuando haya suficientes datos (si no los hay, se producirá probablemente un sobreajuste a los datos de *train*). Al igual que las MFCC, parecen separar bien fricativos, vocales y silencios, sin embargo, por lo parecidas que son estas características con las MFCC, es probable que las RBM no hayan aprendido a obtener características relevantes de las MFCC.

4.5.1. Conclusiones de los extractores de características

Por las representaciones de las características en los distintos apartados, parece ser que los mejores resultados se obtendrán a partir de la combinación de características predefinidas y aprendidas, es decir, las MFCC en combinación con las RBM (Figura 4.8). Las MFCC también muestran un espacio de características con clases razonablemente bien separadas, a excepción de los *stops* (ver Figura 4.1). Las RBM, por otro lado, parecen ser capaces de separar bien los fonemas fricativos de los vocales, pero silencios, nasales y *stops* están esparcidos por todo el espacio de características (Figura 4.6). Aunque se intenta solucionar este problema en las DBN, a la vista de los resultados (Figura 4.7), no se aprecia una mejoría en este aspecto. La tabla 4.1

muestra la opinión (siempre subjetiva) sobre estos extractores para las distintas categorías de fonemas.

Categorías / Extractores	MFCC	RBM	DBN	MFCC-RBM
silencios	Bien	Mal	Mal	Bien
fricativos	Regular	Bien	Bien	Bien
vocales	Bien	Bien	Bien	Bien
stops	Mal	Mal	Mal	Regular
nasales	Regular	Mal	Mal	Regular

Tabla 4.1: Esta tabla muestra como, en nuestra opinión y basada en la representación de unas pocas componentes principales, cada extractor es capaz de diferenciar cada clase de fonema.

A pesar de esto, hay que tener en cuenta que la varianza explicada para las características MFCC y MFCC-RBM es mayor, en parte debido a que estos extractores cuentan con un número menor de características, por lo que estas figuras nos pueden dar una idea aproximada pero no pueden ser concluyentes para decidir el mejor extractor. La conclusión final sólo podrá ser validada al alimentar un clasificador con cada una de estas características y comparar el porcentaje de acierto obtenido con cada una.

4.6. Estudio del contexto en las RNN

En esta sección vamos a demostrar el efecto de lo que denominaremos contexto en las RNN. Se trata de introducir como entradas de la red un número de ventanas a la derecha e izquierda del fonema en estudio. Este trabajo exploratorio permitirá visualizar cómo el contexto de un fonema es importante. Para comprobar cómo de relevante es este contexto para cada tipo de celda (LSTM, GRU y *vanilla*), se va a realizar una prueba comparando los tres tipos de celdas descritas anteriormente. Se entrenarán, por tanto tres RNNs con estos tres tipos de celdas. Todas ellas tendrán la misma estructura y estarán entrenadas con características MFCC. Finalmente, se segmentarán los datos de *test* en cada fonema que contienen y se testeará cada red con cada uno de estos fonemas. Después, en iteraciones sucesivas, se irá añadiendo más ventanas de contexto a cada fonema.

Todas las redes se han entrenado de un modo *frame-wise*, es decir, calculando un fonema por cada ventana de audio. Esto se hace así porque las redes entrenadas con CTC no tienen por qué emitir la salida dentro del tiempo que dura ese fonema. La Figura 4.9 muestra los resultados de esta prueba.

Observando la tendencia del error en función del número de vecinos, sabemos cuánto contexto están guardando las celdas. Se observa que las redes *vanilla* son las que guardan menos contexto, dado que convergen con aproximadamente 7 ventanas de contexto a cada lado del *frame* analizado, seguido de las LSTM con 10 y por último las GRU con unas 13 ventanas de contexto. Hay que mencionar, que cuanto más contexto se le da a cada fonema, menos fonemas hay que cumplan con ese contexto, es decir, tenemos menos datos para testear las redes. Por ejemplo, todos los fonemas menos los del principio y el final de cada frase cumplirán con tener al menos un fonema a cada lado como contexto, pero muy pocos fonemas cumplirán con tener 20 fonemas a cada lado.

Aún así se observa que, en general estas redes requieren de muy poco contexto para dar una salida correcta. Suponemos que esto es así porque las etiquetas a clasificar (es decir, los fonemas) son señales muy cortas en el tiempo y por tanto no es tan relevante la información muy alejada en el tiempo. Sabiendo que cada ventana son 400 *frames* y que el *offset* son 160, las 7 ventanas de contexto equivaldrían a 1520 *frames*, las 10 a 2000 *frames* y las 13 a 2480 *frames*.

Debido a que la variabilidad de la longitud de los fonemas es muy grande no se puede saber a cuánto contexto corresponde, pero se puede concluir que la mayoría de los fonemas caben en este contexto. Es decir, podemos suponer que un fonema (a cada lado) es más o menos lo que necesitan de contexto las redes para ofrecer buenos resultados.

Aún así, las RNN-*vanilla* no son capaces de dar buenos resultados, probablemente debido al *vanishing gradient* (ver sección 2.8.1). LSTM y GRU responden mejor al contexto debido a que tienen implementada puertas relacionadas con la memoria (ver secciones 2.8.3 y 2.8.2). Los resultados de LSTM son mejores con el añadido de que para este problema convergen antes. Esto puede ser debido a las diferencias en cuanto a las puertas en las celdas LSTM y GRU. Por ejemplo, la puerta LSTM cuenta con la *output gate*, que hace que la información de la celda de memoria no tenga por qué ser tenida en cuenta.

4.7. Resultados de los clasificadores

Al entrenar las redes con CTC no es posible calcular comparando etiqueta a etiqueta si la clase predicha coincide con la real. Esto es debido a que CTC no tiene por qué entregar el mismo número de etiquetas en la frase predicha ($\hat{y}$) que en la frase original (y). Por ello, se hace uso de la distancia de Levenshtein (también llamada *edit distance*, ver [7] capítulo 3) que mide el número mínimo de cambios necesarios para que el vector y sea igual a $\hat{y}$. Estos cambios pueden ser: sustituir un fonema de $\hat{y}$ por otro, eliminar un fonema o insertar un fonema. El valor del criterio *accuracy* se define como

$$accuracy = \frac{N_T - S - D - I}{N_T}$$

donde N_T es el número total de fonemas en y, S el número de sustituciones necesarias, D el número de eliminaciones, e I el número de inserciones. Es frecuente encontrar la métrica PER (Phonema Error Rate o ratio de error en el reconocimiento de fonema). La $PER = 100\,\% - accuracy$.

Para nuestro conocimiento, el estado del arte en cuanto a reconocimiento de fonemas con la base de TIMIT se sitúa en un 13.8 % de PER [59] o lo que es lo mismo, un 86.2 % de *accuracy* [60], usando para ello fMLLR [61] como extractores, con celdas GRU modificadas para eliminar la *reset gate*, usar ReLU como función de activación de $\hat{h}$ y aplicar *batch normalization* [62].

4.7.1. Resultados para distintos clasificadores

Para comparar los distintos clasificadores, entrenaremos la red descrita en la Figura 3.9 con las tres celdas de interés (LSTM, GRU y *vanilla*). Como el objetivo de esta sección es comparar el clasificador, sólo utilizaremos las características extraídas con MFCC debido a que son las que menos dimensiones tienen y por tanto el entrenamiento es más rápido. Estos resultados se muestran en la tabla 4.2.

Tabla 4.2: Resultados del reconocimiento de fonemas en TIMIT para la arquitectura propuesta de RNN con distintos tipos de celdas. Se usan MFCCs como características.

Tipo de celdas	Accuracy
LSTM	80.1 %
GRU	73.5 %
vanilla	56.4 %

La Tabla 4.2 muestra que las LSTM son las que mejor resultado ofrecen, seguido por las GRU. Como era de esperar, las redes RNN *vanilla* ofrecen un muy pobre resultado. Para el análisis del resto de características se utilizarán por tanto redes LSTM.

4.7.2. Resultados para distintos extractores de características

Aunque las características extraídas mostradas en las diferentes Figuras (ver Figuras 4.1, 4.6, 4.7 y 4.8) dan una idea aproximada de cómo de bien funciona cada extractor, no se puede asegurar su eficacia hasta que no sean aplicadas sobre un clasificador. Por esto, en esta sección evaluaremos cada uno de los métodos de extracción de características con el mejor clasificador de la sección anterior, en este caso las LSTM. Sólo utilizaremos el mejor extractor porque el tiempo aproximado empleado en entrenar cada clasificador es de unas 15 horas.

Por tanto, se entrenará la misma arquitectura mostrada en la Figura 3.9 con celdas LSTM con cada una de las características explicadas en la sección anterior. Para todas estas redes, se han buscado los mejores parámetros sobre unas pocas épocas, para usarlos luego con el entrenamiento completo. Todas las redes se han entrenado durante 30 épocas. Todas parecen haber convergido en estas 30 épocas.

A la vista de los resultados mostrados en la Figura 4.10, el extractor más adecuado para este problema parece ser MFCC, aunque aún lejos del estado del arte [59].

En cuanto a la RBM, aunque las características aprendidas parecían ser buenas, la representación de éstas en las primeras componentes de PCA ya parecía presagiar que los resultados no lo serían tanto, principalmente por la poca separación que parecía haber entre los fonemas silencio, *stops* y nasales. Esta estructura se ha probado tanto con el reentrenamiento de los *bias* como sin él. El reentrenamiento de éstos parece ayudar notablemente a los resultados de esta red, sin embargo, no parecen ser el extractor más adecuado para este problema.

En cuanto a la DBN, aunque parecía ayudar a los problemas de separación de clases de la RBM, su mejoría al final es casi anecdótica. Esta estructura, sin embargo, tiene muchas posibles opciones de mejora, ya que estos resultados podrían mejorar al variar el número de capas o las activaciones de éstas, entre otras cosas.

Finalmente, las MFCC-RBM, por las características mostradas en la Figura 4.8 parecían presagiar el mejor resultado, sin embargo, aunque no es malo, sigue siendo peor que utilizando sólo un extractor MFCC. Al igual que sucede con las DBN, es posible que variando algunos parámetros se pueda conseguir un mejor resultado.

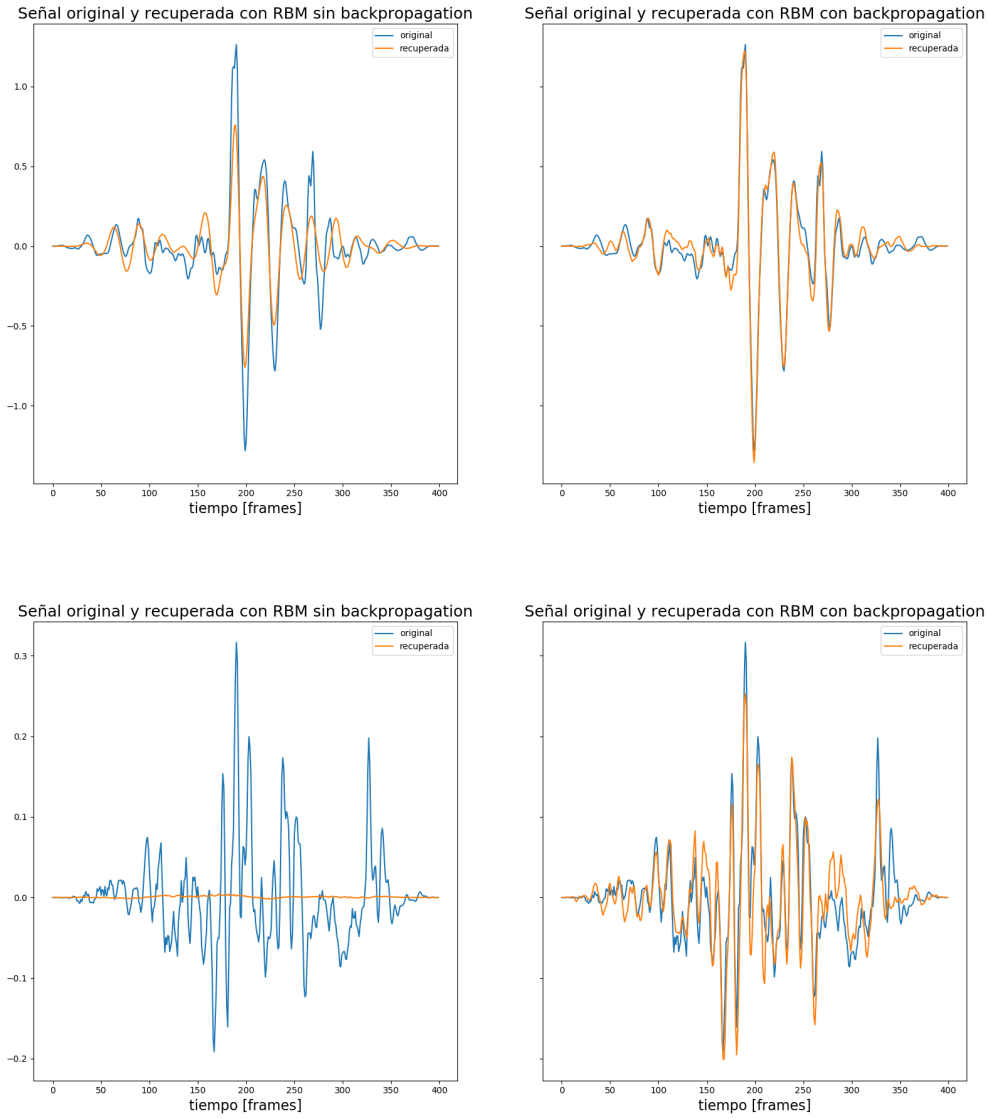


Figura 4.4: Señal recuperada con y sin *backpropagation* en los *bias*. Debido a los bajos valores de los *bias*, en muchas ocasiones la señal recuperada es prácticamente nula. La dos Figuras superiores muestran un ejemplo de la señal en la que la recuperación es buena tanto sin realizar *backpropagation* en los *bias* como aplicándola. En las dos Figuras inferiores, se muestra el caso donde la recuperación sin *backpropagation* es inadecuada y cuando se aplica el reentrenamiento de los *bias*, se puede visualizar que la recuperación de la señal mejora sustancialmente.

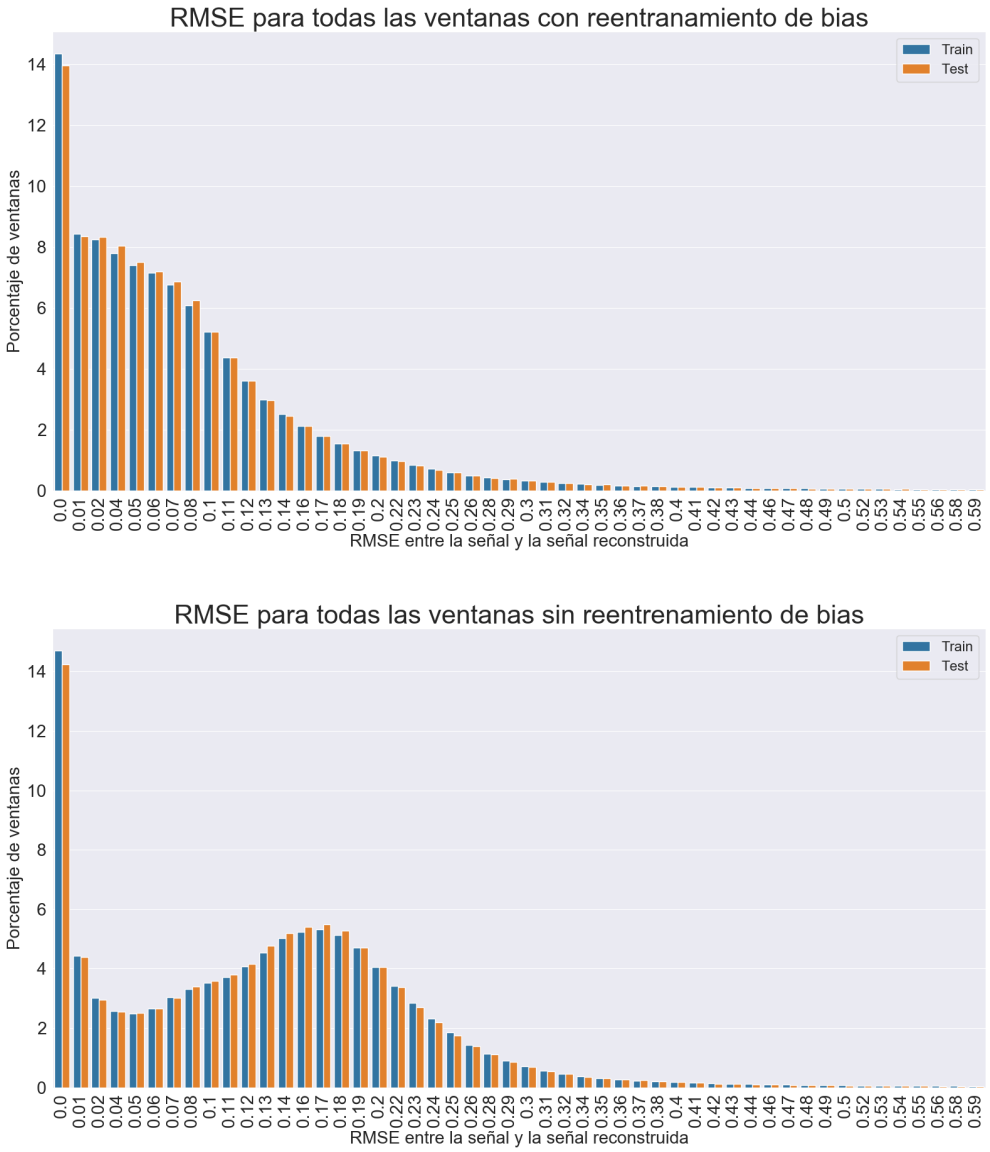


Figura 4.5: Histogramas de los errores medidos con RMSE sobre todas las ventanas de la señal original y la recuperada. El panel superior muestra el histograma de estos errores cuando se corrigen los *bias* con *backpropagation*. El panel inferior, el histograma de estos errores cuando no se corrigen los *bias*.

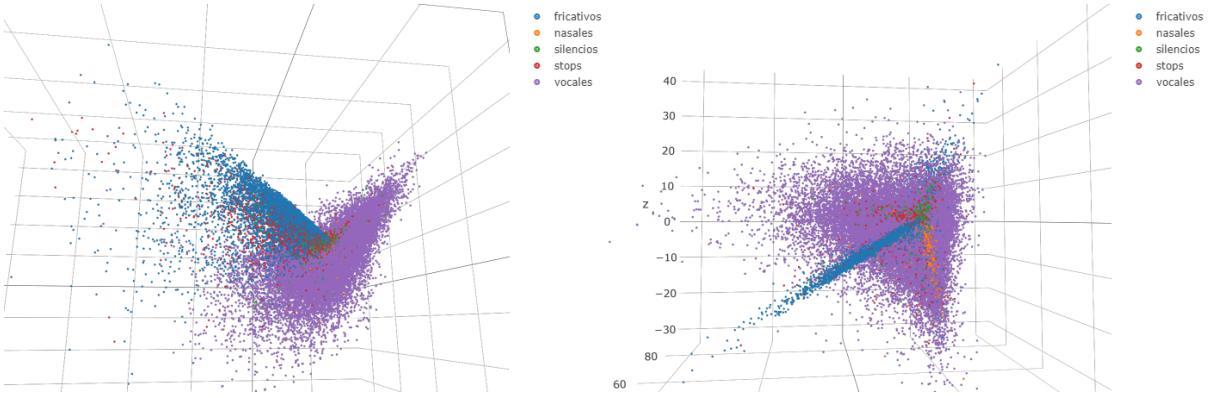


Figura 4.6: Representación de las características extraídas con RBM después de reducir su dimensionalidad con PCA. Cada punto representa una ventana de señal distinta. El panel de la izquierda muestra los componentes 0, 1 y 2 de PCA. Explica el 10.6 % de varianza. El panel de la derecha muestra los componentes 0, 3 y 4 explicando el 7.7 % de varianza.

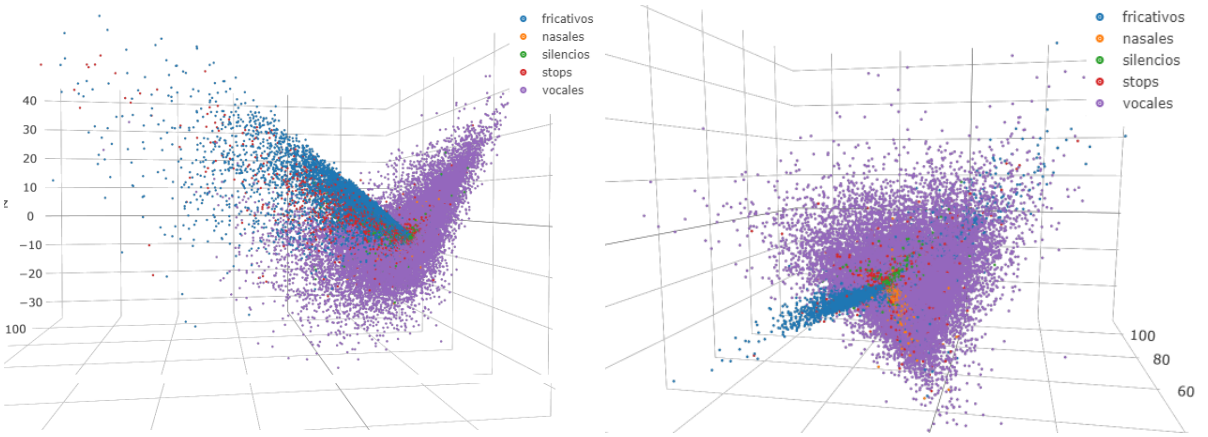


Figura 4.7: Características extraídas con DBN después de reducir su dimensionalidad con PCA. El panel de la izquierda muestra los componentes 0, 1 y 2 de PCA. Explica el 12.4 % de varianza. El panel de la derecha muestra los componentes 0, 3 y 4 explicando el 9 % de varianza.

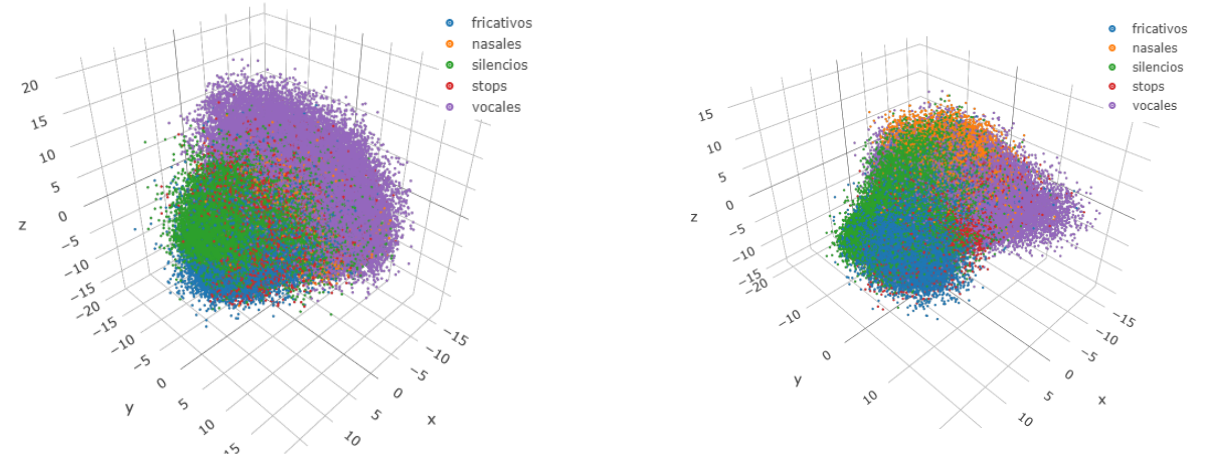


Figura 4.8: Características extraídas de MFCC con una RBM gaussiana-binaria después de reducir sus dimensiones con PCA. El panel izquierdo muestra los componentes 0, 1 y 2 de PCA. Explica el 52.9 % de varianza. El panel derecho muestra los componentes 0, 3 y 4 explicando el 44.13 % de varianza.

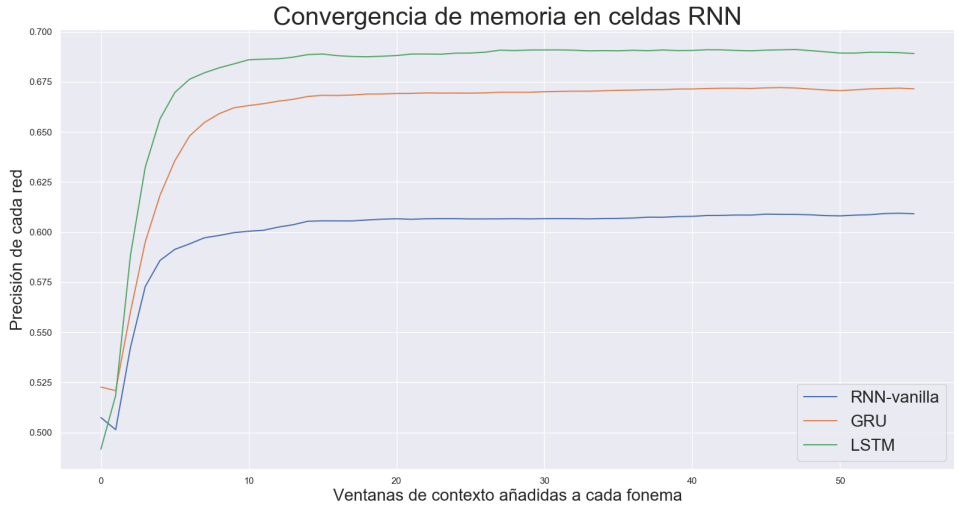


Figura 4.9: Prueba de memoria en RNN. El eje x indica cuántas ventanas de contexto le entran a la RNN. Se puede apreciar en la figura que hay un número de ventanas a derecha e izquierda del fonema a partir del cual la precisión de la red no aumenta significativamente.

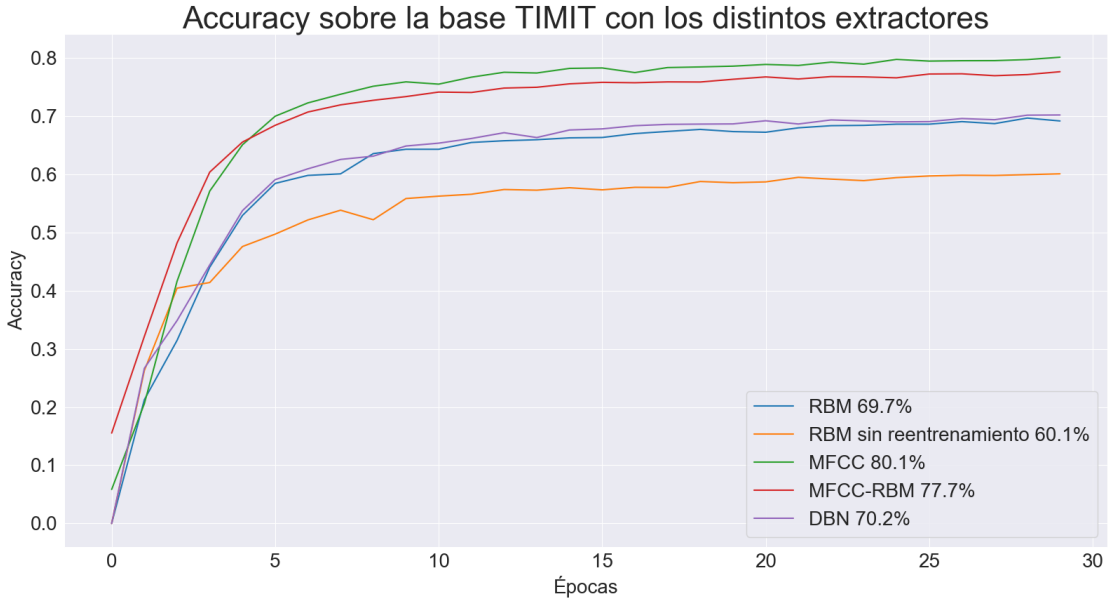


Figura 4.10: Evolución del conjunto de *test* durante el aprendizaje.

5

Conclusiones y trabajo futuro

5.1. Conclusiones

En el capítulo 2 se han presentado, desde un punto de vista teórico, algunas de las técnicas más clásicas para la extracción de características de audio (MFCC), así como otras basadas en aprendizaje no supervisado (RBM). También se han explicado en profundidad algunos de los clasificadores más comunes para trabajar con señales temporales de esta naturaleza. Tanto los HMM como las RNN representan distintos enfoques, más clásico y más moderno, con los que abordar los problemas del reconocimiento de audio.

Todas estas técnicas en su conjunto, representan las partes fundamentales con las que se ha de tratar para iniciarse en la ciencia del *speech recognition*, por tanto, con las explicaciones técnicas del capítulo 2 y los distintos *scripts* disponibles en `https://github.com/isaacgg/TFM/blob/master` se considera el objetivo principal de este trabajo cumplido (ver sección 1.2).

En el capítulo 3, se empieza explorando la base de datos escogida (TIMIT). En el desarrollo de este trabajo se muestra como está estructurada, qué información relevante contienen sus distintos archivos y de qué manera están codificadas las señales. Después, se continúa realizando un análisis exploratorio de los datos del que se obtienen diversas conclusiones interesantes. Demostramos como distintas clases de fonemas comparten características comunes, como en el caso de la corta duración de los *stops* el espectro ruidosos de los fricativos o la alta energía de los vocales. Con este análisis completado, consideramos el subobjetivo de analizar la base de datos concluido.

También en el capítulo 3, definimos los parámetros para los distintos extractores que usaremos para obtener las características relevantes de las señales en TIMIT. Se proponen cuatro extractores: MFCC para la categoría de extractores predefinidos, RBM para la de extractores con aprendizaje no supervisado, DBN como mejora de los extractores con aprendizaje no supervisado, y MFCC-RBM como la combinación de extractores predefinidos junto con no supervisados. Además, se propone una mejora original sobre el entrenamiento de las RBM, consistente en el reentrenamiento de los *bias* que en nuestros experimentos. Con la incorporación de este reentrenamiento se mejora el resultado en la clasificación final casi en un 10 %.

En el capítulo 4 empleamos estos extractores sobre la base de datos para extraer las características y las representamos para extraer conclusiones. De la representación de éstas, observamos

que el extractor predefinido (MFCC) se comporta mejor que los que emplean aprendizaje automático para extraer características discriminantes sobre fonemas. Aunque estos últimos no hayan obtenido los mejores resultados, los extractores que emplean aprendizaje tienen aún así la ventaja de que pueden adaptarse a otros tipos de señales. Esto podría ser de gran utilidad para aquellas señales en las cuales las características predefinidas no consigan buenos resultados. Con todo esto, damos con completado el subobjetivo del estudio y análisis de los extractores.

De nuevo en el capítulo 3, se presenta la arquitectura seleccionada para el clasificador de características. Sobre esta misma estructura, se realiza una prueba para medir el contexto necesario por las redes recurrentes para obtener la mejor clasificación posible en materia de reconocimiento de fonemas. Según los resultados obtenidos, se observa que las redes GRU emplean más contexto fónico que las LSTM para obtener la mejor salida posible, sin embargo, son las LSTM las que obtienen mejores clasificaciones.

Finalmente obtenemos los resultados globales (ver Figura 4.10) sobre la base de datos de TIMIT. A la vista de éstos, concluimos que el extractor predefinido (MFCC) en conjunto con las redes LSTM es lo que mejor se comportan para esta tarea. Con esto, consideramos el último subobjetivo como concluido.

5.2. Trabajo futuro

El posible trabajo futuro en este campo es inmenso. Hay multitud de extractores distintos por probar y muchos otros tipos de clasificadores. Dentro de los muchos extractores existentes que podrían ser de interés, se encuentran los autoencoder, *Linear Predictive coding* [63] o *Wavelet Transform* [64]. Incluso con las DBN existen infinitas estructuras que probar. Típicamente, se podría variar el número de capas, las neuronas por cada capa o incluso el tipo de activación de estas capas, entre otras cosas. Como distintos clasificadores por probar, se encuentran por ejemplo los HMM con alguna de sus variantes o incluso una RNN *end-to-end*, es decir, una red a la que le entraría la señal en crudo y sería esta red la encargada de extraer las características.

Además, debe tenerse en cuenta que todos estos modelos han sido aplicados a un tipo de señal muy concreta, como es el reconocimiento de fonemas. Sin embargo, queda por probar su utilidad con distintas señales de audio o incluso las mismas cuando el objetivo es otro, como puede ser el reconocimiento de palabra.

Por último, basándose en los resultados obtenidos en el estudio del contexto fónico se puede diseñar un entorno en el que el entrenamiento de los diferentes fonemas esté balanceado en cuanto al número de ejemplos por clase.

Bibliografía

[1] P. Scanlon, D. P. W. Ellis, and R. B. Reilly. Using broad phonetic group experts for improved speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, 15(3):803–812, March 2007.

[2] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[3] Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[4] S. Davis and P. Mermelstein. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(4):357–366, August 1980.

[5] David E. Rumelhart and James L. McClelland. *Information Processing in Dynamical Systems: Foundations of Harmony Theory*, page 194–281. MITP, 1987.

[6] J S Garofolo, Lori Lamel, W M Fisher, Jonathan Fiscus, D S. Pallett, N L. Dahlgren, and V Zue. Timit acoustic-phonetic continuous speech corpus. *Linguistic Data Consortium*, 11 1992.

[7] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition, 2000.

[8] Carla Lopes and Fernando Perdigao. Phoneme recognition on the timit database. In *Speech technologies*. IntechOpen, 2011.

[9] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[10] R Sathya and Annamma Abraham. Comparison of supervised and unsupervised learning algorithms for pattern classification. *International Journal of Advanced Research in Artificial Intelligence*, 2, 02 2013.

[11] E. Oran Brigham. *The Fast Fourier Transform and Its Applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.

[12] A.V. Oppenheim, A.S. Willsky, S.H. Nawab, and G.M. Hernández. *Señales y sistemas*. Pearson Educación. Pearson Educación, 1998.

[13] F. J. Harris. On the use of windows for harmonic analysis with the discrete fourier transform. *Proceedings of the IEEE*, 66(1):51–83, Jan 1978.

[14] K. K. Paliwal, J. G. Lyons, and K. K. Wójcicki. Preference for 20-40 ms window duration in speech analysis. In *2010 4th International Conference on Signal Processing and Communication Systems*, pages 1–4, Dec 2010.

[15] William J Williams, Mark L Brown, and Alfred O Hero. Uncertainty, information, and time-frequency distributions. In *Advanced Signal Processing Algorithms, Architectures, and Implementations II*, volume 1566, pages 144–157. International Society for Optics and Photonics, 1991.

[16] Subhadeep Chakraborty. Advantages of blackman window over hamming window method for designing fir filter. 2013.

[17] Prajoy Podder, Tanvir Zaman Khan, Mamdudul Haque Khan, and M Muktadir Rahman. Comparative performance analysis of hamming, hanning and blackman window. *International Journal of Computer Applications*, 96:1–7, 06 2014.

[18] T. K. Roy and M. Morshed. Performance analysis of low pass fir filters design using kaiser, gaussian and tukey window function methods. In *2013 2nd International Conference on Advances in Electrical Engineering (ICAEE)*, pages 1–6, Dec 2013.

[19] S S. Stevens, J Volkmann, and E B. Newman. A scale for the measurement of the psychological magnitude pitch. *Journal of the Acoustical Society of America*, 8:185–190, 01 1937.

[20] E Zwicker. Subdivision of the audible frequency range into critical bands (frequenzgruppen). *Journal of the Acoustical Society of America*, 33(2):248, 02 1961.

[21] X. Huang, A. Acero, A. Acero, and H.W. Hon. *Spoken Language Processing: A Guide to Theory, Algorithm, and System Development*. Prentice Hall PTR, 2001.

[22] M. Narasimha and A. Peterson. On the computation of the discrete cosine transform. *IEEE Transactions on Communications*, 26(6):934–936, June 1978.

[23] Edmund Y Lam and Joseph W Goodman. A mathematical analysis of the dct coefficient distributions for images. *IEEE transactions on image processing*, 9(10):1661–1666, 2000.

[24] Geoffrey E. Hinton and Terrence J. Sejnowski. Analyzing cooperative computation. In *Proceedings of the Fifth Annual Conference of the Cognitive Science Society, Rochester NY*, 1983.

[25] Geoffrey E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Comput.*, 14(8):1771–1800, August 2002.

[26] Robert M. Bell, Yehuda Koren, and Chris Volinsky. The bellkor solution to the netflix prize. 2007.

[27] Asja Fischer and Christian Igel. An introduction to restricted boltzmann machines. In Luis Alvarez, Marta Mejail, Luis Gomez, and Julio Jacobo, editors, *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, pages 14–36, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[28] David E. Rumelhart and James L. McClelland. *Information Processing in Dynamical Systems: Foundations of Harmony Theory*. MITP, 1987.

[29] Geoffrey E. Hinton. A practical guide to training restricted boltzmann machines. In *Neural Networks: Tricks of the Trade*, 2012.

[30] S. Geman and D. Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(6):721–741, Nov 1984.

[31] Geoffrey E Hinton. A practical guide to training restricted boltzmann machines. In *Neural networks: Tricks of the trade*, pages 599–619. Springer, 2012.

[32] Navdeep Jaitly and Geoffrey Hinton. Learning a better representation of speech sound waves using restricted boltzmann machines. pages 5884 – 5887, 06 2011.

[33] Ryan Kiros, J. Adams, Hugo, Rus Nitish, Tanya Amit, G Vinod, James Maks, Marcus Tijmen, Ilya Jonathan, T. Tintu George, Alex Graves Olya, Krizhevsky, Mohammed Schwing, Jimmy Chris, and A Rahman. Exploring deep learning methods for discovering features in speech signals. 2014.

[34] Scott M. Thede and Mary P. Harper. A second-order hidden markov model for part-of-speech tagging. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics on Computational Linguistics*, ACL '99, pages 175–182, Stroudsburg, PA, USA, 1999. Association for Computational Linguistics.

[35] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, 39(1):1–38, 1977.

[36] K. . Lee and H. . Hon. Speaker-independent phone recognition using hidden markov models. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(11):1641–1648, Nov 1989.

[37] R. Gray. Vector quantization. *IEEE ASSP Magazine*, 1(2):4–29, April 1984.

[38] Douglas A. Reynolds. Gaussian mixture models. In *Encyclopedia of Biometrics*, 2009.

[39] Farheen Fauziya and Geeta Nijhawan. A comparative study of phoneme recognition using gmm-hmm and ann based acoustic modeling. 2014.

[40] L. Li, Y. Zhao, D. Jiang, Y. Zhang, F. Wang, I. Gonzalez, E. Valentin, and H. Sahli. Hybrid deep neural network–hidden markov model (dnn-hmm) based speech emotion recognition. In *2013 Humaine Association Conference on Affective Computing and Intelligent Interaction*, pages 312–317, Sep. 2013.

[41] Shanshan Han, Minfei Zhang, Penglin Li, and Jinjie Yao. Svm-hmm based human behavior recognition. In Qiaohong Zu, Bo Hu, Ning Gu, and Sopheap Seng, editors, *Human Centered Computing*, pages 93–103, Cham, 2015. Springer International Publishing.

[42] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323, 2011.

[43] Ingo Lütkebohle. Gradients for an rnn. `https://github.com/go2carter/nn-learn/blob/master/grad-deriv-tex/rnn-grad-deriv.pdf`, 2017. [Online; accedido el 08-Junio-2019].

[44] F. A. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: continual prediction with lstm. In *1999 Ninth International Conference on Artificial Neural Networks ICANN 99. (Conf. Publ. No. 470)*, volume 2, pages 850–855 vol.2, Sep. 1999.

[45] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[46] Rui Fu, Zuo Zhang, and Li Li. Using lstm and gru neural network methods for traffic flow prediction. In *2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC)*, pages 324–328. IEEE, 2016.

[47] Kazuki Irie, Zoltán Tüske, Tamer Alkhouli, Ralf Schlüter, and Hermann Ney. Lstm, gru, highway and a bit of attention: An empirical overview for language modeling in speech recognition. In *Interspeech*, pages 3519–3523, 2016.

[48] Tensorflow webpage. `https://www.tensorflow.org/`. [Online; accedido el 08-Junio-2019].

[49] Keras webpage. `https://keras.io/`. [Online; accedido el 08-Junio-2019].

[50] Theano documentation. `http://deeplearning.net/software/theano/`. [Online; accedido el 08-Junio-2019].

[51] Pytorch webpage. `https://pytorch.org/`. [Online; accedido el 08-Junio-2019].

[52] Floydhub webpage. `https://www.floydhub.com/`. [Online; accedido el 08-Junio-2019].

[53] T. Jeff Reynolds and Christos A. Antoniou. Experiments in speech recognition using a modular mlp architecture for acoustic modelling. *Inf. Sci.*, 156(1-2):39–54, November 2003.

[54] Violin plot. Technical report, National Institute of Standards and Technology (NIST), October 2015.

[55] Edouard Grave, Armand Joulin, Moustapha Cissé, Hervé Jégou, et al. Efficient softmax approximation for gpus. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1302–1310. JMLR. org, 2017.

[56] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376. ACM, 2006.

[57] Awni Hannun. Sequence modeling with ctc. *Distill*, 2(11):e8, 2017.

[58] Jonathon Shlens. A tutorial on principal component analysis. *Educational*, 51, 04 2014.

[59] Mirco Ravanelli, Philemon Brakel, Maurizio Omologo, and Yoshua Bengio. Light gated recurrent units for speech recognition. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2(2):92–102, 2018.

[60] Mirco Ravanelli, Titouan Parcollet, and Yoshua Bengio. The pytorch-kaldi speech recognition toolkit. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6465–6469. IEEE, 2019.

[61] Sree Hari Krishnan Parthasarathi, Bjorn Hoffmeister, Spyros Matsoukas, Arindam Mandal, Nikko Strom, and Sri Garimella. fmllr based feature-space speaker adaptation of dnn acoustic models. In *Sixteenth annual conference of the international speech communication association*, 2015.

[62] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[63] KR Aida-Zade, C Ardil, and SS Rustamov. Investigation of combined use of mfcc and lpc features in speech recognition systems. *World Academy of Science, Engineering and Technology*, 19:74–80, 2006.

[64] Z Tufekci and John N Gowdy. Feature extraction using discrete wavelet transform for speech recognition. In *Proceedings of the IEEE SoutheastCon 2000.'Preparing for The New Millennium'(Cat. No. 00CH37105)*, pages 116–123. IEEE, 2000.